(2)

BDM

BDM INTERNATIONAL, INC.
7915 JONES BRANCH DRIVE
McLEAN, VIRGINIA 22102-3396
(703) 848-5000

AD-A242 920

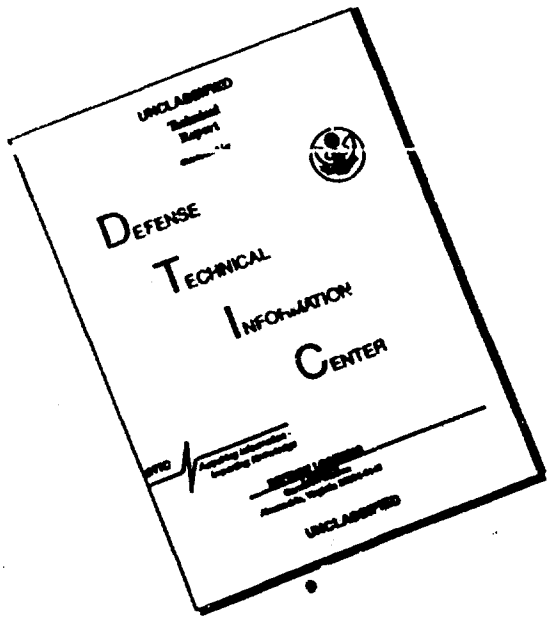
AFOSR-TR- 91 0821

DTIC
S
C

[redacted]

AF
N
T
(AFSC)
and is
IAW 190-12

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

BDM INTERNATIONAL, INC.
7915 JONES BRANCH DRIVE
McLEAN, VIRGINIA 22102-3396
(703) 848-5000

ANALOG OPTICAL NEURAL NETS:

A NOISE SENSITIVITY ANALYSIS

FINAL TECHNICAL REPORT

September 11, 1991

BDM/MCL-91-0364-TR

Sponsored by
Air Force Office of Scientific Research
Under Contract #F49620-89-C-0115
Technical Monitor: Dr. Alan Craig, AFOSR/NE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	9/11/91	Final Tech Report - 7/21/89 - 7/20/91

4. TITLE AND SUBTITLE
Analog Optical Neural Nets: A Noise Sensitivity Analysis

5. FUNDING NUMBERS
C:
F49620-89-C-0115

6. AUTHOR(S)
Dr. Michael W. Haney
Mr. James J. Levy
Dr. Ravindra A. Athale

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
BDM International, Inc.
7915 Jones Branch Drive
McLean, VA 22102-3396

8. PERFORMING ORGANIZATION REPORT NUMBER
BDM/MCL-91-0364-TR

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
USAF, AFSC
Air Force Office of Scientific Research
Building 410
Bolling AFB, DC 20332-6448

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The development of analog optical implementations of neural networks such as the multilayer perceptron with learning by backward error propagation (BEP) requires an understanding of the noise sensitivity of such architectures. In this program computer simulations were used to study the effects of component and system noise on the performance of such optical implementations. A hybrid optical/electronic parallel architecture capable of both the forward pass and backward error propagation steps of training data presentation was conceived and modeled. The simulations showed that the most significant effects were due to the nonlinear response of the spatial light modulators used to store and update the neural weights. Another conclusion of the simulation results was that increasing the hidden layer size increases noise immunity significantly.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Optical Neural Nets, Opto-electronic noise	121
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE.	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
unclassified	unclassified	unclassified	unlimited

Abstract

Neural networks represent a promising alternative to traditional AI approaches. The development of analog optical implementations of neural networks such as the multilayer perceptron with learning by backward error propagation (BEP) requires an understanding of the noise sensitivity of such architectures. The objective of this program is to study the effects of component and system noise on the performance of such optical implementations. The method used is computer simulation.

In this first phase of the program, the one-hidden layer perceptron with back propagation was simulated using a simplified, device-independent noise model. The results point to a distinct noise threshold above which the learning mechanism is corrupted. The efficiency of learning based on variations within back propagation and on the initializing method was also studied.

In the next phase, device-dependent noise models were used. To this end a hybrid optical/electronic parallel architecture capable of both the forward pass and backward error propagation steps of training data presentation was conceived and modeled. The simulations showed that the most significant effects were due to the SLMs' nonlinear response. The "noise" processes studied in the simulations included the SLM nonlinear response, SLM drive circuit noise, nonuniform SLM response, finite SLM contrast ratio, optical crosstalk, photodiode shot and thermal noise, and CCD shot and thermal noise. It was found that the SLMs' nonlinear response significantly increases the number of cycles needed for convergence in learning. Several solutions to this problem were characterized in the simulations. Another conclusion of the simulation results was that increasing the hidden layer size increases noise immunity significantly.

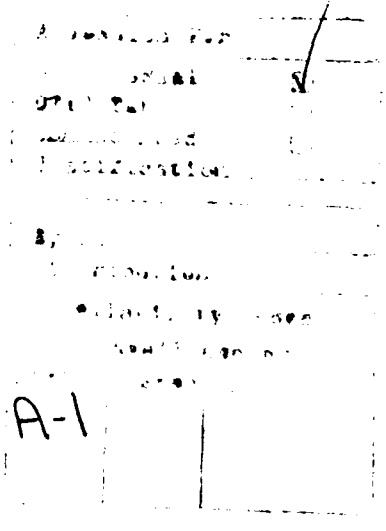


Table of Contents

Abstract			iii
Table of Contents			iv
I.	Background		1
	A.	Program Motivation	1
	B.	Program Goal	3
II.	Taxonomy of Neural Networks and Training Methods		4
	A.	Categories of Neural Network Models and Model Selection Philosophy	4
	B.	The Backward Error Propagation Model	6
	C.	Exemplar-based Model	6
III.	Simulations of Back Propagation		8
	A.	Description of Tool	8
	B.	Backward Error Propagation Details	8
		1. Gradient Descent	10
		2. Initial Conditions	12
		3. Creating Target Vectors for Classes	14
	C.	Summary of Results in Phase I	16
IV.	Optical Architecture for BEP		18
	A.	Overview	18
	B.	Individual Imperfections	22
		1. The SLMs	28
		2. The Optical Imaging System	39
		3. The Detectors	41
	C.	Combinations of Key Effects	47
		1. Temporal Noise and Malus's Law in the Weights	47
		2. Crosstalk and Shot Noise	48
		3. The Complete Detector Noise Models	49
		4. The Complete, Compensated SLM Noise Model	50
	D.	Hidden-layer Redundancy	50
		1. The SLMs	54
		2. The Optical Imaging System	62
		3. The Detectors	64
	E.	Combinations of Key Effects (J = 8)	72
		1. Temporal Noise and Malus's Law in the Weights	72
		2. Crosstalk and Shot Noise	73

3. The Complete Detector Noise Models 74
4. The Complete, Compensated SLM Noise Model 75
F. The Entire Architecture 77
V. Exemplar-based Model 81
A. The Nearest-neighbor Network 81
B. The Nestor Learning System 82
C. Modified BEP 84
VI. Discussion and Conclusions 89
VII. References 91
VIII. Key Personnel 93
IX. Conference Presentations 94
Appendix: Program Listings 95
A. Straight Back Propagation 95
B. Optical Back Propagation 101
C. Nestor Learning System 111
D. Modified BEP 114

I. BACKGROUND

A. Program Motivation

Neural networks are computers that are based on organizational and functional principles of information processing systems found in nature, like the brain and retina. Hence, neural nets consist of many analog processing elements that are densely interconnected and they are applied to problems such as sensor pre-processing, pattern recognition/classification, and motor control. In neural networks, long-term information is stored as interconnection weights which signify the efficacy of interaction between neurons. Transient information is represented by the neural outputs, which, for a given neuron, is a nonlinear function of its state of activation. Each neuron's state of activation is determined by a number of factors: its external inputs, which are the weighted outputs from other neurons; its previous states of activation; and other specific and nonspecific global signals. The values of the interconnection weights change more slowly than the neuron's state of activation. The gradual evolution of the interconnection weights has been widely postulated as the primary learning mechanism that makes animals adapt to a constantly changing environment. The proposed neural net models for problem solving attempt to emulate these intriguing characteristics of biological information processing systems. References 1, 2, and 3 contain representative examples of recent research on neural net development.

The primary advantage of adaptive neural net problem solving approaches over conventional methods is that the networks learn how to solve problems semi-autonomously; from labeled or unlabeled training data, the network learning rules calculate weights which will produce the appropriate outputs. Thus, there is no need for standard artificial intelligence techniques like investigation of the nature of the problem and extensive programming of solution strategies—all that is required is access to raw data. This approach is especially useful in several scenarios: when designing systems that can be applied to a variety of problems with little modification; when the size and complexity of the problem makes rule discovery by hand prohibitively expensive; when rapid solution to a problem is desired; when the nature of the problem is dynamic; or when it is difficult to ascertain the structure of the problem due to noisy and/or distorted data.

Nonadaptive neural networks are designed to solve specific problems. With this approach, the appropriate set of weights and initial conditions are determined by the user. Later, when inputs are presented, the state of the network converges to the proper answer through the network dynamics. In these networks, either extensive calculations must be performed to find the appropriate set of weights or a detailed prior knowledge of the desired processing is needed. The network must, therefore, be used many times to justify the cost of its construction.[4] Hence, this type of network is most suitable for sensor pre-processing type applications where the same operations must be performed on many data sets.

Optical systems have been proposed as candidates for neural network implementation for a number of reasons. Foremost are the analog and parallel nature of neural computation—optical systems have been employed for a number of years to solve significant problems, like synthetic aperture radar imaging and RF spectrum analysis, with parallel, analog hardware. In addition, neural nets often require complex and dense interconnections between the neurons—and interconnection and communication are the primary advantages optics has over electronics. Finally, neural nets require analog storage of interconnection weights that can be accessed and updated in parallel and several 2-D and 3-D optical devices exist that can provide this functionality. [5,6]

For analog optical numeric processors, the accuracy of the overall computation is strongly dependent on the accuracy of the analog optical devices. When such numeric processors were first proposed, the computations considered for analog optical systems were primarily linear (matrix operations) and the accuracy of devices was quite low; therefore, so was the accuracy of the overall processor. Hence, the available applications were limited to those requiring low precision. This motivated the investigation of analog optical systems for neural nets, which were postulated to require low accuracy computation. The first neural network that optics researchers chose to implement, the Hopfield model,[7,8] was indeed relatively insensitive to inaccuracies in the response of the analog components. As it turns out, the very features of the Hopfield model that make it relatively insensitive to inaccurate components, namely the particular type of distributed and redundant storage/computation, also limits its storage capacity, and hence, its utility.

Since the publication of the Hopfield model, there has been a number of potentially useful neural network models reported in the literature along with proposals

for their optical implementation.[9] However, it has been commonly assumed that since the Hopfield model was relatively impervious to hardware imprecision, so are these other models. This is not necessarily true—each model must be examined closely to determine its own sensitivity to analog hardware imprecision. Since many modern neural net models improve upon the storage capacity of the Hopfield model by means of less redundant storage techniques, they may lead to systems which are more sensitive to noise. Dependence of the noise sensitivity on the size of a given neural net model is another issue of great concern since the neural net approach to problem solving may become competitive with conventional approaches only when the network is very large. It is clear from the above discussion that the detailed study of a neural network when implemented in any analog technology (optical or electronic) is of critical importance.

B. Program Goal

The main goal of the research program was to study the effects of component and system noise on the performance of optical implementations of selected neural net models. The noise could be due to variations in the response of different components, finite accuracy in controlling signal amplitudes, or signal- and time-dependent noise due to quantum and thermal fluctuations. Since the size and speed of neural net processors will be factors that influence their utility, the results of this study will be critical in determining the ultimate applicability of optical neural nets. The identification of those parts of the neural net model that are particularly sensitive to system noise will stimulate investigations into new techniques of data representations and formatting to increase the robustness of the models. Similarly, the identification of the limiting devices or materials in the optical implementation of the selected neural net models will lead to exploration of different technological and architectural alternatives for improved performance.

II. TAXONOMY OF NEURAL NETWORKS AND TRAINING METHODS

A. Categories of Neural Network Models and Model Selection Philosophy

Neural network models can be designed to perform several operations, e.g., associative memory, optimization, filtering, pattern classification. The performance criteria for a given model are determined by its intended application. We choose neural pattern classifiers for further investigation under this program. For these models one performance criterion is simply the number of training or test patterns the model misclassifies. Another is corruption of the learning curve during training. The effect of system noise on the selected neural net classifiers is investigated using the above-mentioned criteria.

Neural net models can be categorized according to several different parameters. The first one of these parameters is the topology of the neural net (single or multiple layers of processing elements). The second one is the processing element response (linear, hardclipping nonlinear, or continuous nonlinear). The third parameter deals with the selection of learning algorithm. Error-driven algorithms can be used with labeled training data consisting of input patterns along with their correct classification. Unlabeled training data uses self-organizing algorithms capable of autonomously clustering the input patterns into distinct classes and adjusting the internal parameters of the neural net to generate the desired classification.

The limitations of a single layer neural net model in classification have been well documented.[10] Hence we have selected a multilayer neural net classifier. It can be readily seen that a linear processing element response reduces a multilayer neural net to an equivalent single layer neural net thereby suffering from the same limitations. Therefore we have chosen a nonlinear (hardclipping or continuous) response for the processing element. For the purpose of this study we choose the error-driven learning algorithms that are used with labeled training data. The self-organizing systems were not selected because optical implementations for them are relatively less developed.

Figure 1 depicts a taxonomy of multilayer feedforward neural networks. As shown, each algorithm is spedific to layout (architecture type) and training philosophy.

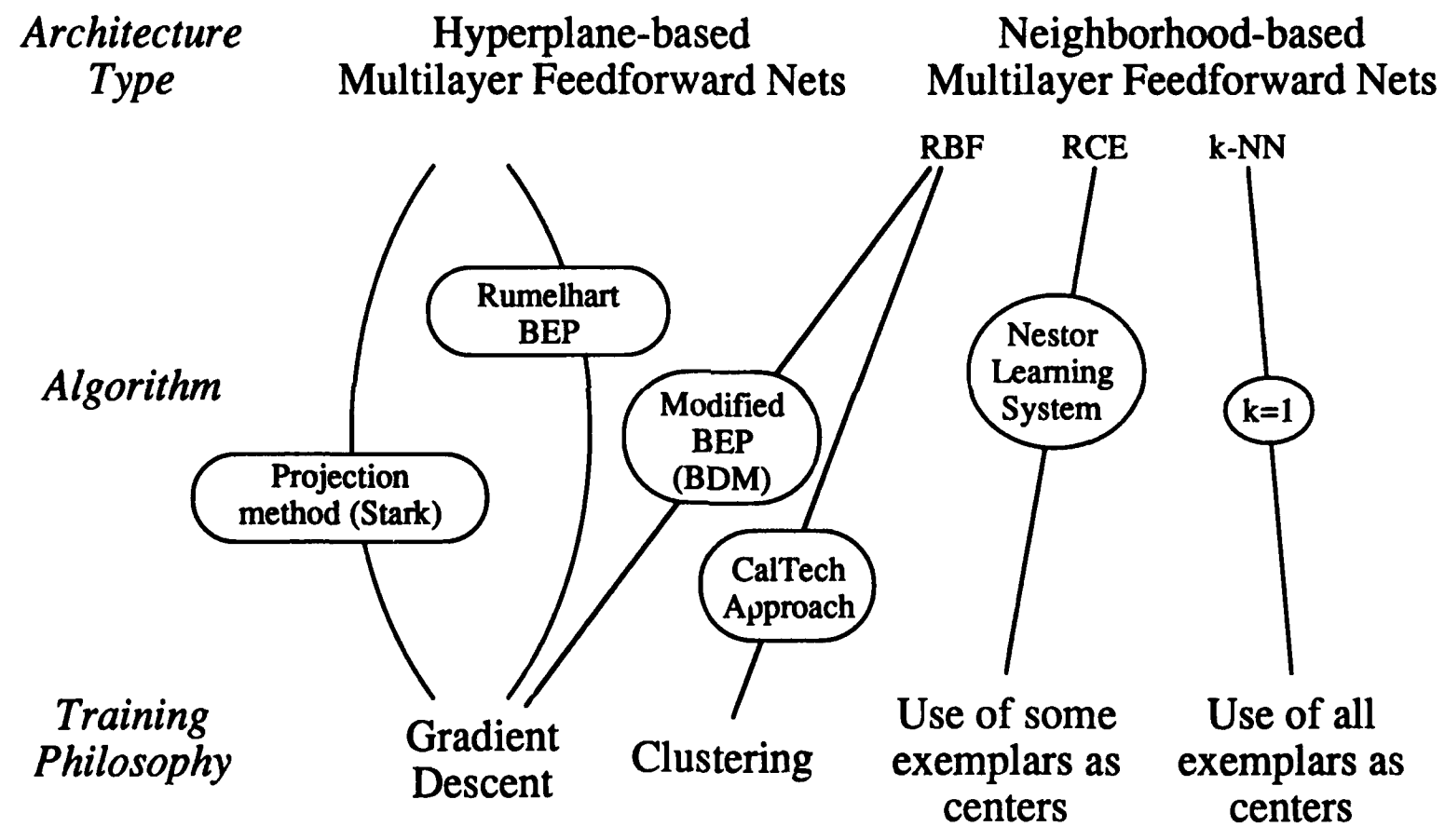


Figure 1. A taxonomy of multilayer feedforward neural networks.

B. The Backward Error Propagation Model

The most prominent multilayer, nonlinear, error-driven neural net is the multilayer perceptron trained by the method of backward error propagation (BEP).[11] This is a least mean-square error approach that uses a gradient descent algorithm. The internal parameters of the neural net (the connection weights and processing element thresholds) are modified such that the new weights lead to a decrease in the total mean-square error. This model is a direct descendant of the Widrow-Hoff Adaline model that was developed for a linear, single layer neural net model.[12] It has also been successfully applied to interesting problems such as distinguishing underwater man-made objects from natural ones based on their sonar returns[13] and solar flare prediction.[14]

The BEP model uses a continuously differentiable nonlinear response for the processing element. Therefore the signals propagating between different layers are fully analog. This could lead to error accumulation. This feature makes the issue of system noise particularly relevant to the BEP model and hence appropriate for this study. Fully optical implementations of BEP model have been proposed.[15] Hybrid optical-digital electronic systems have also been proposed in which part of the training procedure is performed in an auxiliary digital electronic processor.[16,17] The effect of weight quantization on the learning performance of a BEP model has been previously reported.[17,18] These studies were geared towards specific optical or electronic implementations in which the connections were stored electronically in a digital representation. The current study extends that work to quantify the effects of analog system noise in the weights as well as in the processing element computation.

C. Exemplar-based Model

The BEP classifier is characterized by a long training interval in which the training pairs have to be presented as many as several thousand times before the correct weight vectors are identified High computational accuracy may therfore be needed to achieve convergence. This learning procedure also can have the tendency to get stuck in a local minimum for the total error function. On the other hand, the number of processing elements (and hence the total weight storage requirement) is independent of the *number* of exemplars, i.e., training input patterns. The BEP learning procedure, thus, makes efficient use of resources, in that the number of interior (or hidden)

processing elements needed is a function of the decision space complexity, rather than simply the quantity of training data.

Exemplar-based classifiers typically require many hidden processing elements, but can be trained in a relatively short time and with low accuracy. These tradeoffs suggest that the exemplar-based classifiers may be more suited to optical technology that provides large storage capacity but poor computational accuracy. For this reason, we have been examining exemplar-based classifiers for further study. Optical processors can be restricted to the first or second layer of the neural net performing the computationally intensive operations. High precision analog or digital electronics can be used in the final layers that perform the classification.

III. SIMULATIONS OF BACK PROPAGATION

A. Description of Tool

Throughout the program, we have striven to understand the dynamics of a multilayer perceptron which is learning by back propagation of errors, and to observe the effects of the various noise sources on these dynamics.

To these ends we have been using PC-Matlab as a programming and analysis tool. PC-Matlab works with 1-D and 2-D variables. When it is running an ".m" file (a program), it can display in text or graphical form the ever-changing state of the network. Information such as a learning curve can be stored in a ".mat" file.

The core of our simulation program is found in two modules, "uinit.m" and "uctrain.m." These modules are written specifically for the case of one hidden layer, although it is simple to modify them to add layers. Among other tasks, "uinit.m" loads the training data, initializes the weights and biases, and sets up the beginning of a training session. Then "uctrain.m" iteratively updates the weights, tracks the mean-square error, and provides for early termination of training upon fulfillment of a convergence criterion. Note that a training seed (trseed) is used with Matlab's random number generator, so that an identical set of "random" noise spikes may be used in different runs, if desired.

These two program modules run as though the hidden layer size, initializing criteria, the learning rate, and the various noise values are already defined in PC-Matlab's work space. A ".m" file called "uframe.m" prompts for these data so that "uinit.m" and "uctrain.m" may be run without causing undefined-variable errors.

These three ".m" files, as well as some of the nested ".m" files, are listed in the Appendix.

B. Backward Error Propagation Details

To provide a deeper understanding of the simulation tools, we shall now describe exactly how back propagation proceeds, giving due attention to the updating procedure. Back propagation is defined for a multilayer perceptron, a neural network containing

nodes configured in an input layer, a number of hidden layers, and an output layer. A one-hidden-layer architecture is shown in Figure 2. (Strictly speaking, the input layer units are not full-fledged nodes. They simply broadcast the input signals to the nodes in the hidden layer. Therefore, many call this a *two*-layer network.)

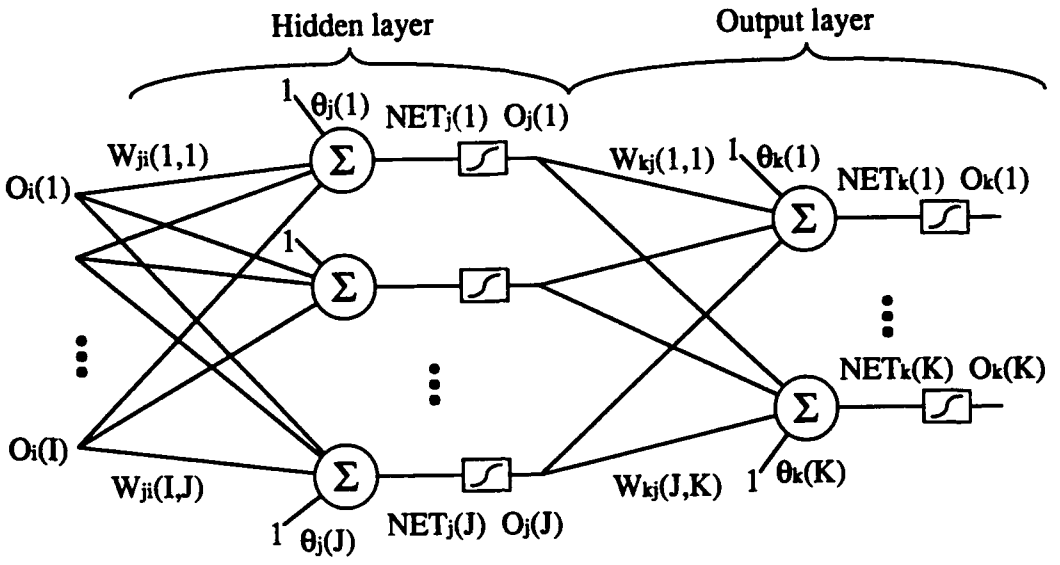


Figure 2. A perceptron with one hidden layer. The network has I inputs (i = 1 to I), J hidden processing units, and K output processing units. The weight $W_{ji}(i,j)$ refers to the connections strength from input unit i to hidden unit j, and similarly at the next layer. Note that the inputs to the biases are held at 1. After summing, a given unit performs a sigmoidal nonlinearity.

The multilayer perceptron learns iteratively, each iteration having two main steps. In the first of these, it processes the inputs via what is commonly referred to as a forward pass. The input elements are multiplied by the weights; then they are summed; this sum is often called the internal activation. It is then transformed ("thresholding") by a nonlinearity in the hidden nodes. The outputs of the hidden nodes are processed in similar fashion. The inputs and outputs (o terms) are unipolar, due to the thresholding; in some cases, the inputs may be restricted to binary values. The weights and biases are bipolar and continuous.

In equation form, the forward pass is written:

Forward Pass

1. $Net_j = \sum_i o_i W_{ji} + \theta_j \qquad o_j = 1/(1 + \exp(-Net_j))$

2. $Net_k = \sum_j o_i W_{kj} + \theta_k \qquad o_k = 1/(1 + \exp(-Net_k))$

The two equations on the right express a particular type of thresholding, namely, a sigmoidal nonlinearity.

1. Gradient Descent

Learning in this context refers to the modification of the weights and biases in a network. For the network to perform properly, all layers are required to learn. Gradient descent is a training algorithm which iteratively updates the weights and biases in a given layer based on its inputs, outputs, and target outputs. That is, the weight update is supposed to move the weights in a direction in weight space in which the mean-square error over all training pairs decreases the most. This is expressed mathematically as a reduction in the mean-square error computed over all components of all training pairs:

$$\Delta W \propto -\partial E/\partial W_{kj} \text{ where } E = \sum_r \sum_k (t_{rk}-o_{rk})^2$$

t_{rk} and o_{rk} being the target vector's and output vector's kth element, respectively, for the rth training pair. This is often called the delta rule. The derivative can be computed by application of the chain rule:

$$\partial E/\partial W_{kj} = (\partial E/\partial Net_k) \times (\partial Net_k/\partial W_{kj}).$$

The quantity $\partial Net_k/\partial W_{kj}$ is just o_j. The other link is called $-\delta_k$, and is itself expressible as a chain:

$$\partial E/\partial Net_k = -\delta_k = (\partial E/\partial o_k) \times (\partial o_k/\partial Net_k)$$

$$=-(t_k-o_k) \times (o_k\,(1-o_k))$$

Note that $o_k\ (1-o_k) = \partial(1/(1 + \exp(-Net_k)))/\partial Net_k$; it's the derivative of the sigmoid nonlinearity. That is why that particular threshold function is chosen: because it is continuous and easily differentiable.

During presentation of an input (forward pass), the values t_k and o_k are readily available at the output layer; t_k is one element of the training data. In the hidden layer, these terms become t_j and o_j, of which the the latter is the actual output of the jth hidden unit upon presentation of training input r. The former, t_j, is the jth element of what is called the internal representation; this is not known *a priori*.

Back propagation[11] represents the first successful method for calculating an equivalent of the target-minus-output error for hidden layers. This error term is calculated by propagating the output layer's error term backward through the output layer's weights. That is, it can be shown that

$$\partial E/\partial o_j = -\sum_k \delta_k W_{kj}$$

The steps within back propagation are thus summarized:

Backward Error Propagation

3. $\delta_k = o_k\ (1-o_k)\ (t_k-o_k)$
4. $\delta_j = o_j\ (1-o_j) \sum_k \delta_k W_{kj}$
5. $\Delta W_{kj} = \eta\ o_j\ \delta_k \qquad \Delta\theta_k = \eta\ \delta_k$
6. $\Delta W_{ji} = \eta\ o_i\ \delta_j \qquad \Delta\theta_j = \eta\ \delta_j$

In the PC-Matlab modules, o_i is a row vector of length I; Net_j, o_j, θ_j, and δ_j are row vectors of length J, and similarly for k. The variable η is called the learning rate.

The derivation, as presented in Reference 11, leaves some questions open. For example, are the weights and biases to be updated after presentation of each training pair (updating by pattern)? Or are the weight changes to be accumulated in a separate register over the entire epoch of training pairs and then added to the weights (updating

by epoch)? True gradient descent is based on updating by epoch, but some researchers advocate pairwise updating as a means to improved performance.

Recall what gradient descent means: the weights change in the *direction* of greatest decrease in the total mean-square error. True gradient descent requires infinitesimal weight changes. For larger weight changes, a decrease in mean-square error is not guaranteed. An error landscape may be convoluted, and an update may move the weights too far, to a point where the error is higher. In other words, a large learning rate can create oscillations in the plot of mean-square error vs. epoch. A convoluted landscape possesses local minima, which can halt convergence, with the network stuck in an unsolved state.

These problems are often remedied by incorporating a momentum term α which introduces a component of the previous weight change into the current one. Step 5 above becomes

$$\Delta W_{kj}(n) = \eta o_j \delta_k + \alpha \Delta W_{kj}(n-1)$$
$$\Delta \theta_k(n) = \eta \delta_k + \alpha \Delta \theta_k(n-1)$$

and similarly for the hidden layer, where n tabulates the actual update, whether it was pairwise or not. According to Gilbert,[18] momentum is used in updating by pattern to incorporate information about the previous pair, making training based on more complete information.

2. Initial Conditions.

The convergence behavior produced by the back propagation algorithm depends on the initial values of the weights and biases. If all weight values start out equal, the algorithm keeps them so, and the network will not learn. Rumelhart's solution is to initialize the weights and biases with small random values, to provide symmetry breaking. He does not state what "small" means, however.

Is there another way to initialize the weights? Our desire is a configuration which helps the network to solution, but is not specific to any one problem (does not "cheat"). Sheldon Gilbert[18] proposes one method based on Lippmann's[19] discussions on internal representations.

A network which has learned XOR is shown in Figure 3. Figure 4 shows the output of the upper, $j = 1$, hidden unit (vertical axis) as a function of the two inputs (horizontal axes). The magnitude of the input weight vector determines the steepness of the output—how close the "hill" of Figure 4 is to being a step function. The associated decision line, a one-dimensional hyperplane, is just the intersection of this output with the plane $o_j(1) = 0.5$. Each hidden unit in Figure 3 is labeled with a diagram of its decision region, showing this decision line. The arrow indicates in which half-plane the output is greater than 0.5. Note that the dividing decision lines actually pass through the decision region.

Gilbert's method starts with initially random weights and biases, and scales the weight vectors to a uniform magnitude. Then it adjusts the biases so that the dividing hyperplane passes through (0.5, 0.5,...,0.5)—the middle of the decision region. This is illustrated in Figure 5. Next, the weights to the output layer are all set to $1/J$, so that, regardless of the size of J, the output units start out with reasonably-sized Net_k's. For example, even if the initial $o_j = [1, 1,...,1]$, each Net_k term (with $\theta_k = 0$) will be only $J \times (1 \times 1/J) = 1$, well within the linear region of the sigmoid defined in Equation 2. (In point of fact, we have initialized the θ_k terms to small random values within [-0.5, 0.5] in our simulations.) In the simulations that Gilbert performed on 2-D problems, the initializing algorithm went on to force the dividing lines to span 360°. We chose to omit this latter restriction for two reasons: first, it seems too "forced" for typical problems, and second, it encourages deciding which dividing lines should run which way in input space, the answer to which is best found by already knowing the final state of the solved net.

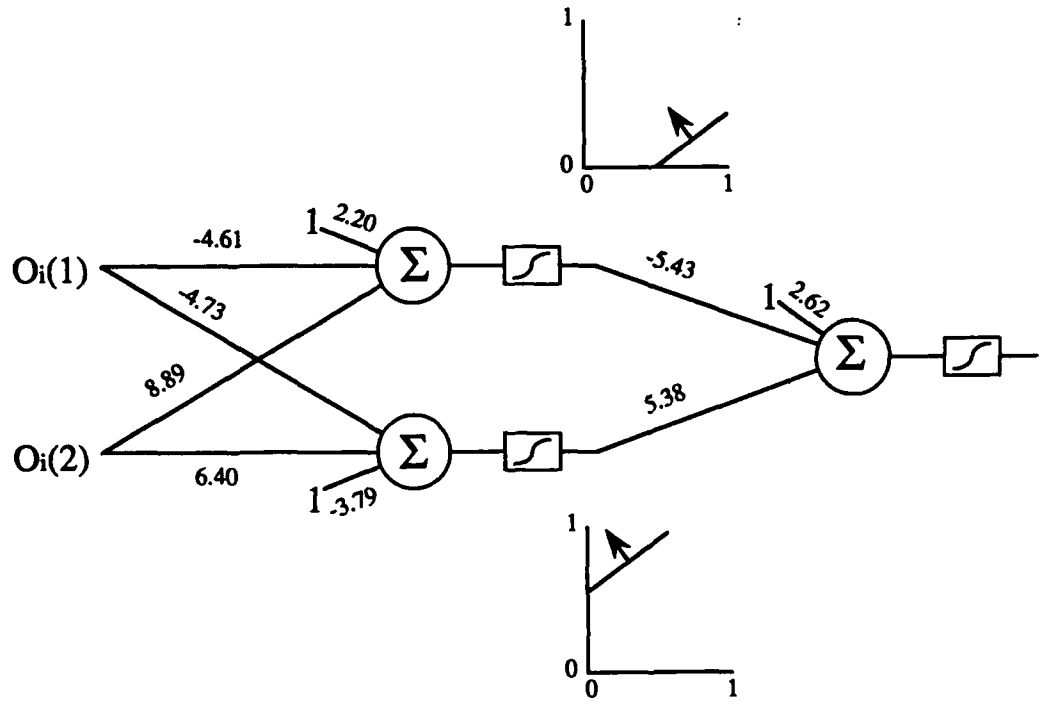


Figure 3. A $J = 2$ solved XOR configuration.

3. Creating Target Vectors for Classes

For classification problems, one can express the classes one of two ways. The first way uses a dedicated output node for each class. A typical training pair has an input vector and an output vector with all elements low except for that corresponding to the correct class. The second way uses a binary representation for each class. Here a four-class problem could be implemented with two instead of four output nodes.

In addition, how the output training vector expresses "high" and "low" is important. Since the activation function asymptotically approaches 0 and 1, using these in the training vector may cause excessively large error signals to be propagated back.

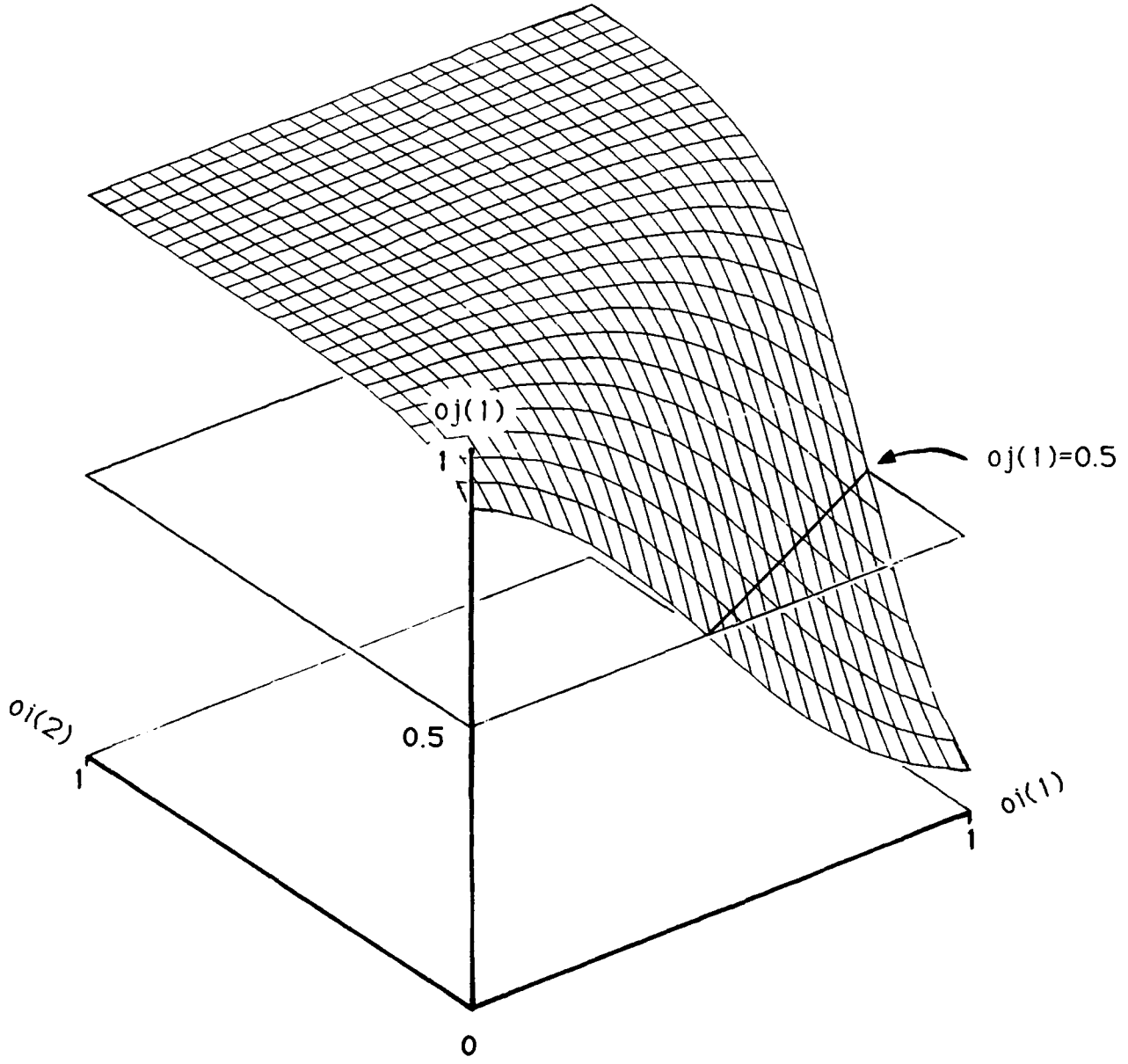


Figure 4. The output of the j = 1 hidden unit as a function of the inputs to the XOR network.

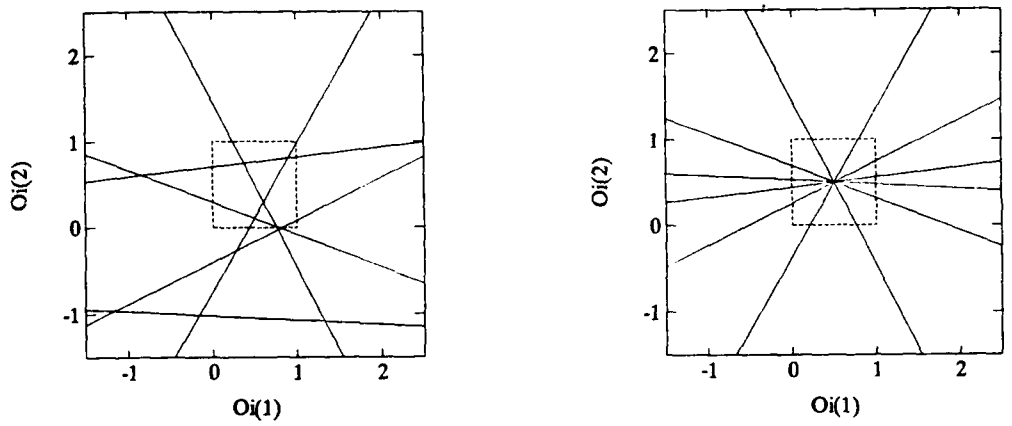


Figure 5. Initial dividing hyperplanes (lines, in this case) before and after application of Gilbert's method.

All of our simulations were two-class problems, where we used $K = 1$ (binary representation). Also, unless otherwise noted, we trained our output on {0.1, 0.9} rather than {0, 1} to avoid the excessive error signals.

C. Summary of Results from Phase I

Earlier Matlab program modules enabled us to run simulations of the back propagation algorithm using a simplified additive noise model. In that model, small Gaussian noise terms were added to all of the weights immediately after they were updated. We also explored a variety of updating by pattern in which hidden weight changes were based on propagating the errors through output weights which were already updated. Although it is quite afield from true gradient descent, this layered updating by pattern converged the fastest. However, updating by epoch was found to be less noise sensitive.

One must also choose a suitable learning rate, large enough to speed performance without oscillations through the error landscape. Oscillations and entrapment in local minima can be avoided using the momentum term α, but as we shall see, this does not lend itself well to optics. It is also wise to choose targets away from the asymptotic tails of the threshold function.

Performance is dramatically improved by initializing the weights so that the dividing hyperplanes created by the hidden units cross the center of the input vector space, and normalizing the weight vector magnitudes.

IV. OPTICAL ARCHITECTURE FOR BEP

A. Overview

The simulation of a true optical neural net must incorporate more than simply additive noise applied to the weights and biases after update. It is necessary to take into account noise introduced by modulators, the optical imaging system, and detectors. This requires designing an architecture capable of handling the whole process, including the forward pass, error back propagation, accumulation of the ΔW's over the epoch, and addition of them at the end. The architecture need not be *optimum*, only plausible.

Our proposed architecture uses optics to perform operations of $O(N^2)$, where N is the typical layer size, e.g., I, J or K; other operations may take place in the electrical domain. The inputs and weight/bias matrices are assumed to be implemented as spatial light modulators (SLMs).

Figure 6 shows our multilayer perceptron architecture for a net in which $I = 3$, $J = 4$ and $K = 2$. SLMs are represented as unshaded planar regions; detectors are shown shaded. Thick arrows represent the propagation of information-carrying light; thin arrows refer to signals in the electrical domain. For simplicity, the figure does not show the necessary cylindrical and spherical imaging lenses, nor the switchable birefringent wave plates needed to direct light the proper way using the polarizing beamsplitters. Wherever possible, we have striven to use the same weights in backward as in forward passes. This saves hardware and reduces noise accumulation.

As depicted in Figure 6, the architecture is performing a forward pass. Recall the governing equations presented in Section III. B. 1. The forward passes are vector-matrix multiplications shown in Figure 6 by black left-to-right arrows. While the inputs o_i and o_j are unipolar and restricted to the interval [0, 1], the weights and biases are bipolar and can have larger magnitudes. We have tracked the weights and biases through simulations on 2-D and 5-D problems and seen values as large as 25. Since the SLMs do not amplify, we let them express values W_{ji}/h, where h corresponds to half the actual range of the weights. (For example, if weights range from -25 to 25, h is set to 25.) Also, we have divided the matrix elements into positive and negative subelements. The Net_j and Net_k are also so divided.

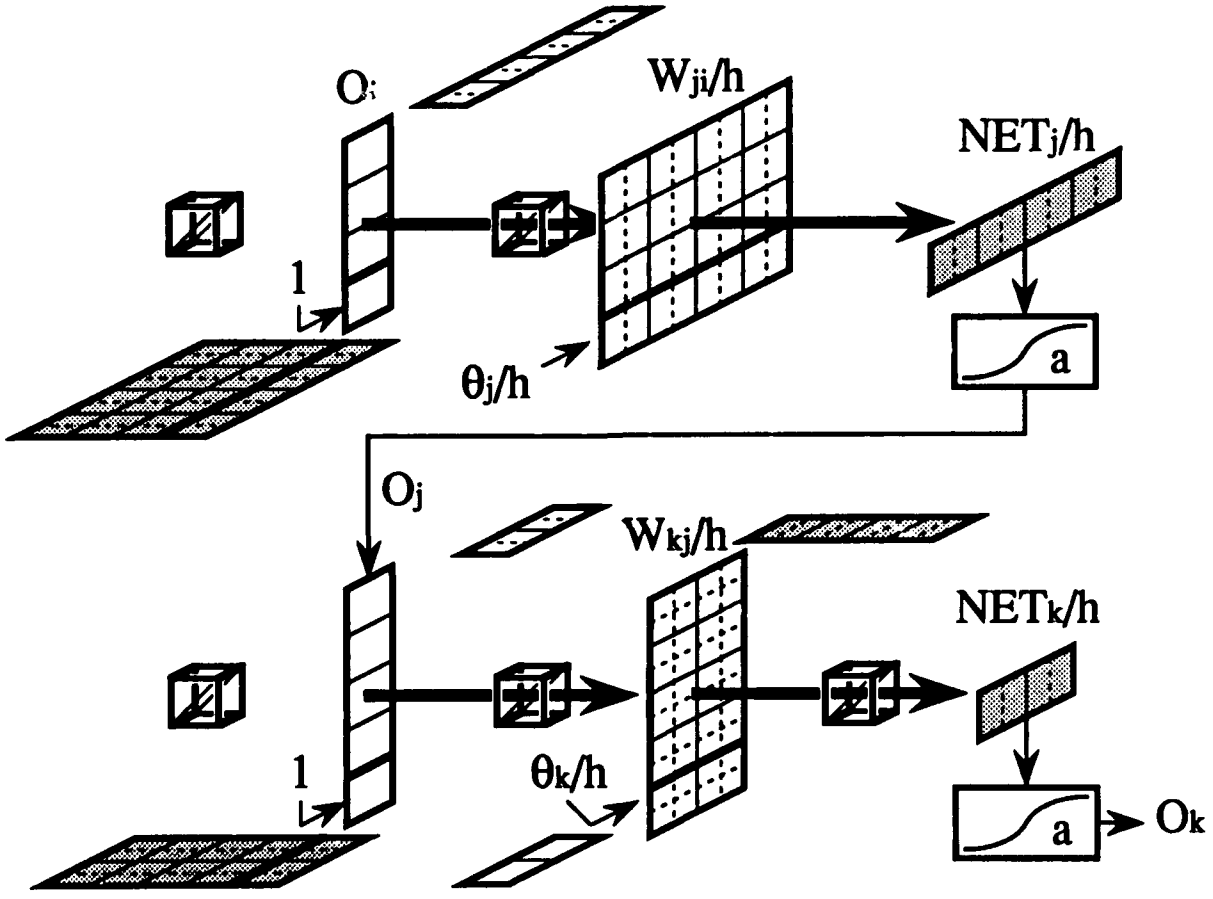


Figure 6. The proposed multilayer perceptron architecture for $I = 3$, $J = 4$ and $K = 2$, performing a forward pass.

The subtraction of $Net(-)$ from $Net(+)$ is performed electronically. The thresholding elements incorporate sigmoids which are a steep, thus performing the operation

$$o_j = 1/(1 + \exp(a \times -Net_j/h)) = 1/(1 + \exp(-Net_j))$$

when $a = h$, and similarly for o_k. The constant a is similar to h; we will examine it more closely below.

Figure 7 shows the same architecture, now performing backward propagation of errors. The error term δ_k is computed electronically (lower right corner of Figure 7). Note that δ_k is bipolar and usually small. In the Phase I simulations, the largest element never exceeded 0.16. To better utilize the full range of the 1-D SLMs, we multiply δ_k by a constant b. Note that $\delta_k b$ is assigned to two SLMs; in one, the values of $\delta_k b$ are sign encoded with (+) and (-) subelements. This SLM is to be used for computing the outer product as in Equation 5 of the governing equations. The outer product between this and the unipolar o_j results in a bipolar ΔW_{kj}, in which the elements are divided horizontally into (+) and (-) subelements. This is compatible with the encoding format of W_{kj}; hence, the weight update process is simplified.

The other SLM to which $\delta_k b$ is assigned is for calculating the vector matrix multiplication inside Equation 4. Since both δ_k and the weight matrix W_{kj} are bipolar, this computation is more difficult to implement than the forward pass. This requires dividing the W_{kj} elements *vertically* into (+) and (-) subelements as shown. As depicted by the gray arrow in Figure 7, the operation

$$W_{kj}^T/h \times \delta_k b$$

occurs in two passes, one for each sign of δ_k. In the first pass, $\delta_k(+)$ is presented; each element of the receiving detector will evaluate two terms, positive and negative. In the second pass, when $\delta_k(-)$ is presented, the formerly positive subelement now receives a negative term

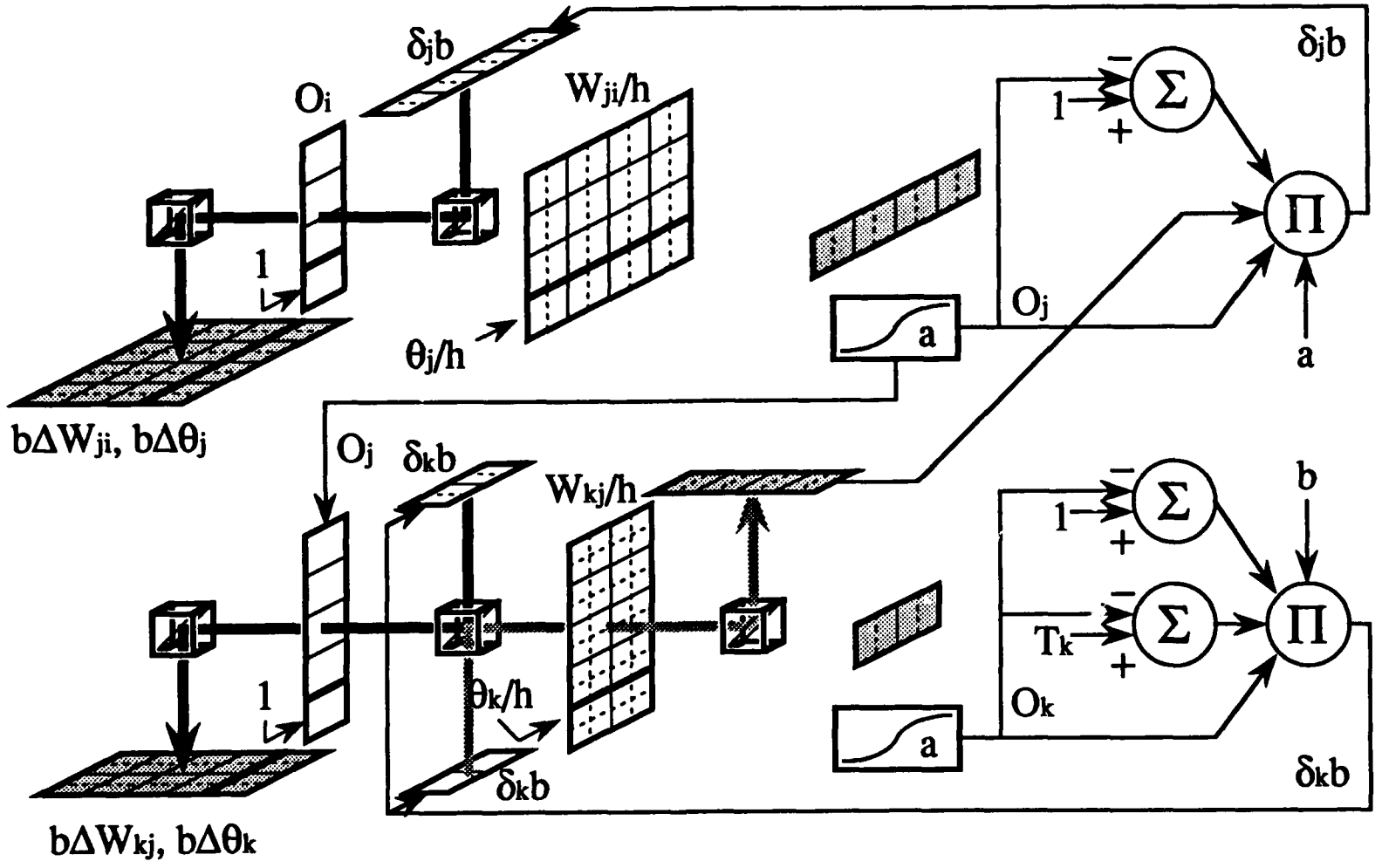


Figure 7. The proposed multilayer perceptron architecture performing backward error propagation.

$$(\delta_k b(-) \times W_{kj}{}^T / h(+))$$

—and vice versa. The receiving detector has associated electronic circuitry to handle this sign interchange.

When this vector matrix multiplication is complete, δ_j (from the rest of Equation 4) can be computed.

With δ_k and δ_j computed, all that remains is two outer products (Equations 5 and 6). These use the light paths shown in black arrows. The receiving detectors are CCDs and perform accumulation over the epoch by time integration. Here the operation multiplies bipolar-by-unipolar, as in the forward passes.

At the end of the epoch, the accumulated ΔW_{ji} and ΔW_{kj} contain (+) and (-) terms, all nonzero. For each weight, the difference between its (+) and (-) terms is computed; then that subelement of the proper sign is set to that value, the other being set to zero. Finally, all the changes are to be added to the weight matrices themselves, electronically. The reason these 2-D operations are allowed to be electronic is that they occur only once per epoch and therefore do not represent a bottleneck.

The Phase II simulations incorporated opto-electronic noise caused by the various transductions in the architecture. In addition to random noise such as shot and thermal noise, the effects of fixed pattern noise, as from nonuniformity and limited contrast ratio, were included. In the next section we take up the mathematical nature of these imperfections and demonstrate their impact on the convergence properties of the algorithm.

B. Individual Imperfections

A complete simulation accounts for all elements that may, according to good engineering judgement, present a potential impediment to the normal convergent behavior of the algorithm. The system described above consists of four subsystems: the sources, the SLMs, the imaging optics, and the detectors. Of these, only the source imperfections have been left out of our work. Since the input vectors to all operations are themselves implemented using row SLMs, the entire system could be illuminated with a common, external, ideal laser source. Compared to the imperfections present in

the other three subsystems, we believe that the laser source does indeed behave ideally, emitting an essentially constant, collimated beam.

For the body of simulations we designed a relatively simple problem we refer to as 2-D skewed corners. The problem requires two input nodes, one output node, and four hidden units to solve. Figure 8 shows the 25 training data used in the training. The figure also shows a typical back propagation-induced solution, in the form of a contour plot. We used Gilbert's method to speed convergence; the steepness constant was set to 4. We also chose η to be 0.45; this was the largest observed value that was not prone to inducing oscillation. We set the magnitude range a at 20; the simluation automatically increases a to make up the loss of dynamic range that occurs whenever a bias is used to smooth the uncompensated weight transfer functions. As for the constant b, we observed in a trial run that the δ_k and δ_j terms reached as high as 0.3. We therefore decided that it was safer simply to leave b at unity.

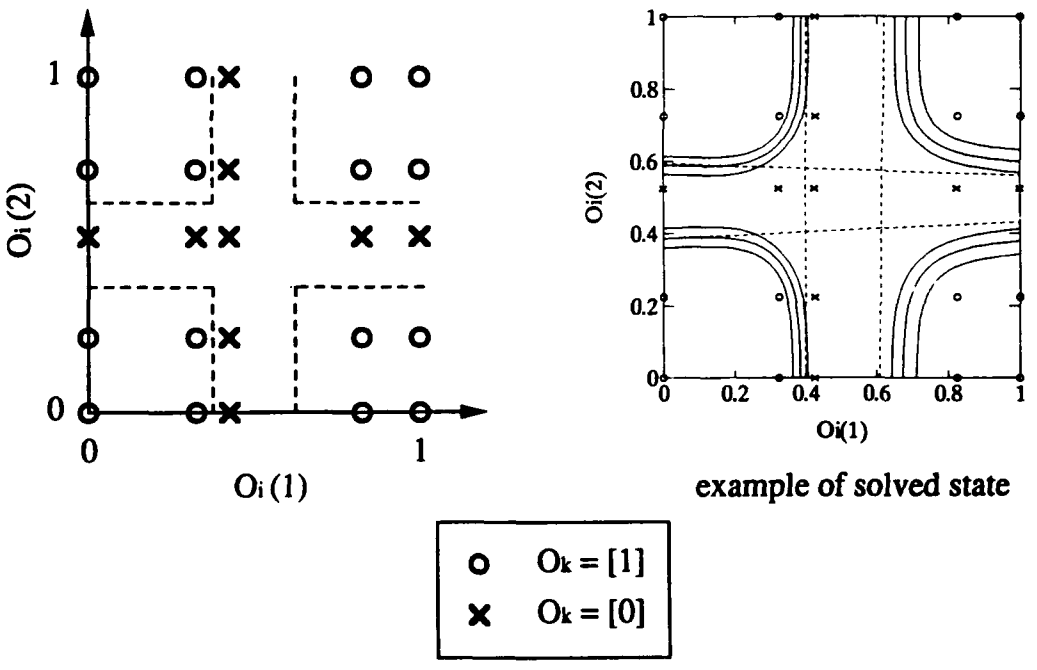


Figure 8. The training set used for most of the simulations. Also shown is a contour plot for a converged network, with output values at 0.3, 0.5, and 0.7. The dotted lines in this plot represent decision boundaries set by each hidden unit.

We chose four representative sets of initial conditions. Using straight back propagation, the four representative cases take 1010, 1400, 1810, and 1950 (rounded to the nearest 10) epochs (or cycles) to solve, respectively. Their learning curves are shown in Figure 9. Figure 10 shows the same learning curves on individual pairs of axes, superimposed with plots of the number misclassified. (Strictly speaking, there is no relationship between the units of mean-square error and the number misclassified, despite their being shown on the same ordinate.) Unless otherwise stated, we let each trial run to $N = 3000$ epochs, regardless of convergence. We also set the simulation to record its progress every DNth epoch, with $DN = 10$, to save memory.

At this point some remarks concerning determination of time to convergence are in order. As the simulation runs, it creates a long row vector called *ldetmse* (an abbreviation for "less detailed mean-square error"). Every DNth epoch, the program appends to *ldetmse* an element which is the mean-square error averaged over the last DN epochs. It also at that time appends to another long row vector, *noff*, which is the number of training pairs misclassified at that time (not an average, unlike *ldetmse*). Generally, the time to convergence is determined by finding the largest n value for which *noff* is nonzero. In Matlab, this is done as follows:

```
x=0:DN:n; max(x.*(noff~=0))
```

For simulations without temporal noise, this metric is adequate. However, temporal noise in any element usually manifests itself as spikes in graphs of both *ldetmse* and *noff*. Consider, for example, the plot of *noff* shown in Figure 11. In this case, *noff* remains spiked until it reaches zero at about $n = 2400$, and then stays at zero with only an occasional spike to one or two at some higher values. The above metric would then yield that convergence had been reached at $n = 2850$, which happened to be the last spike generated. This is not useful; our metric should yield an answer close to 2400 for us to compare performance between this and other simulation runs. Our solution is to divide *noff* into groups of ten elements (100 cycles each for $DN = 10$), take the average over the ten elements, and find the largest n for which this number is above 0.2. In Matlab, the coding reads, for $n = 3000$,

```
xb=100:100:3000;
max(xb.*(mean(reshape(noff(2:301),10,30))>0.2))
```

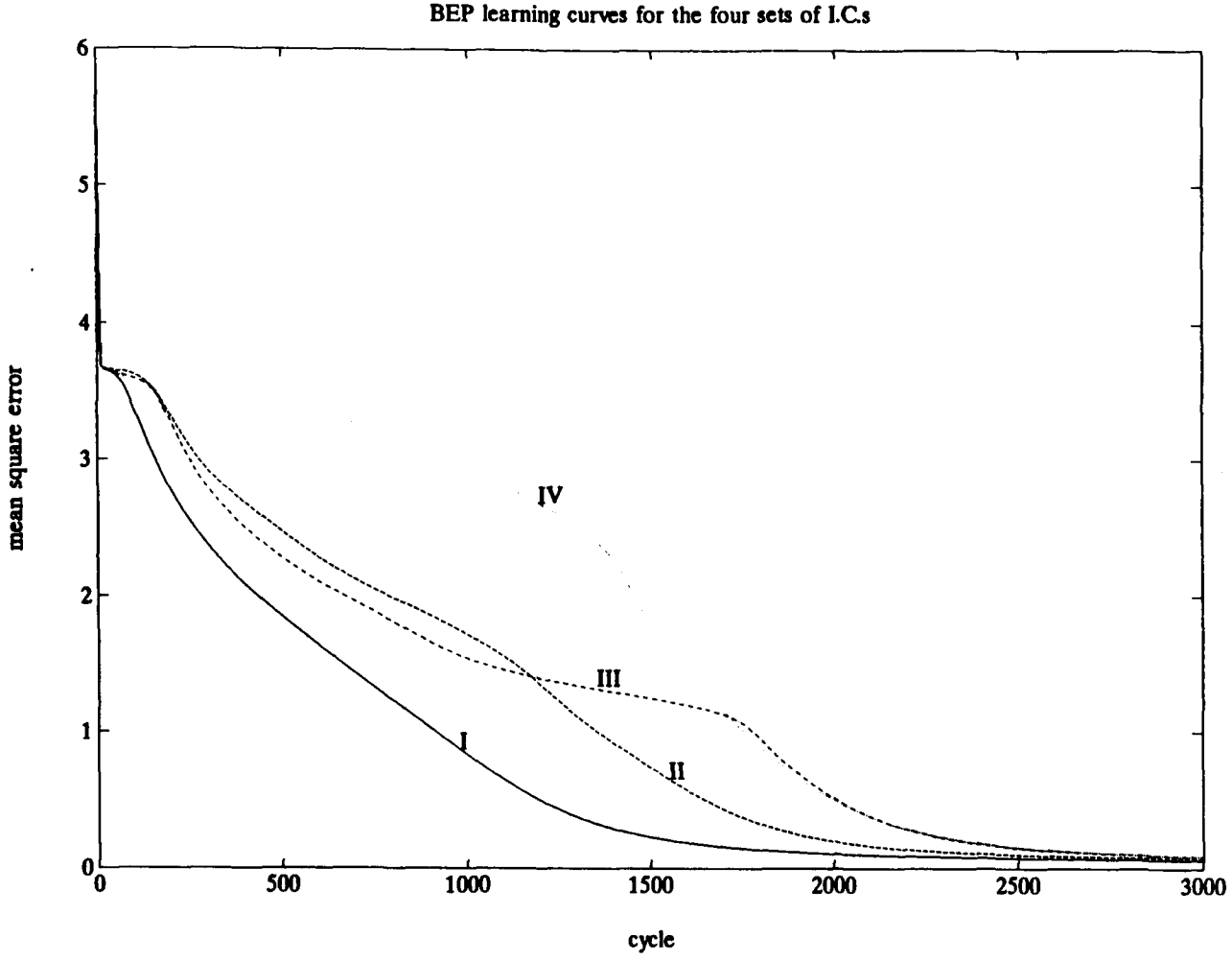


Figure 9. Four representative learning curves generated by back propagation on the training data of Figure 7.

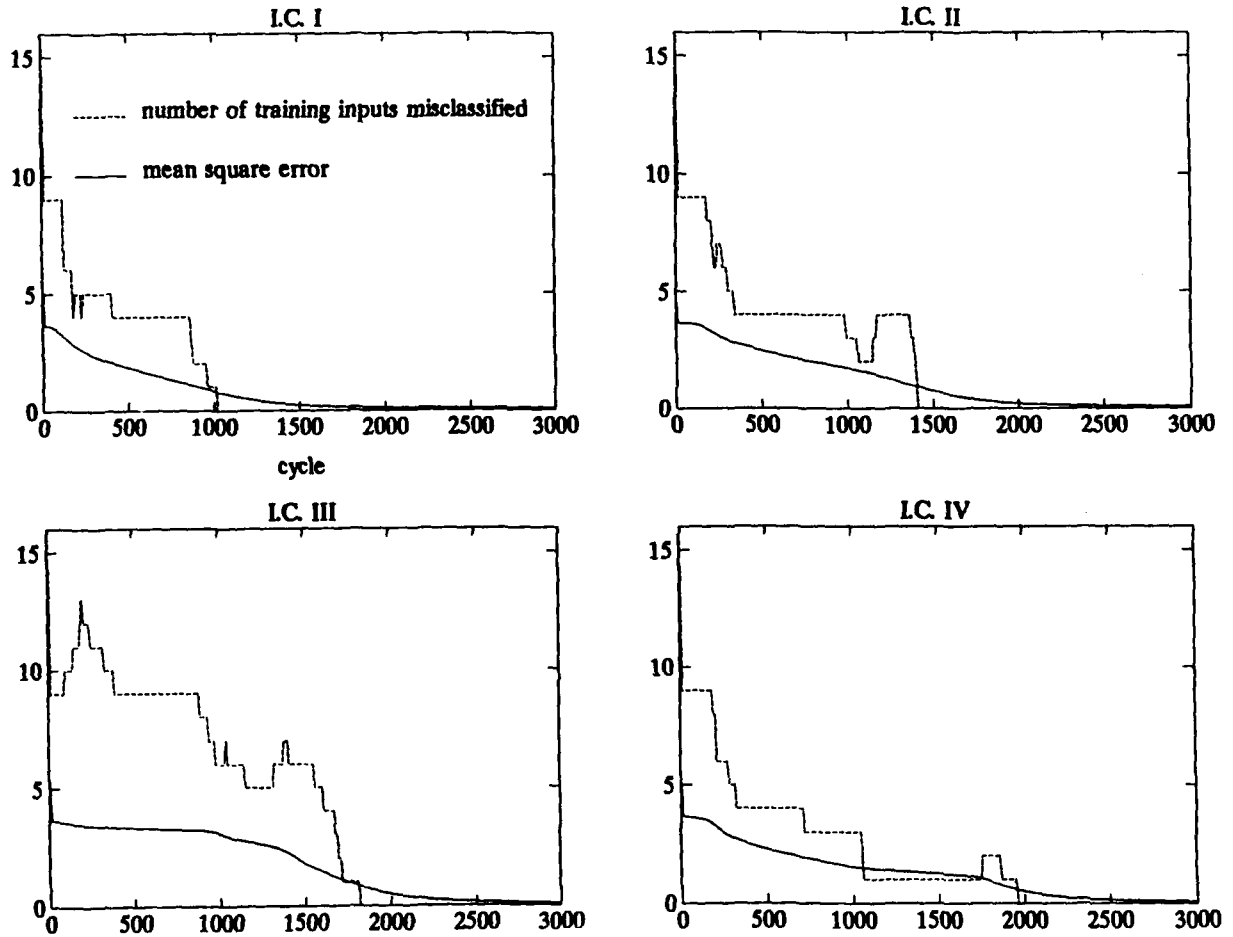


Figure 10. The four representative learning curves along with plots of the number of training data misclassified.

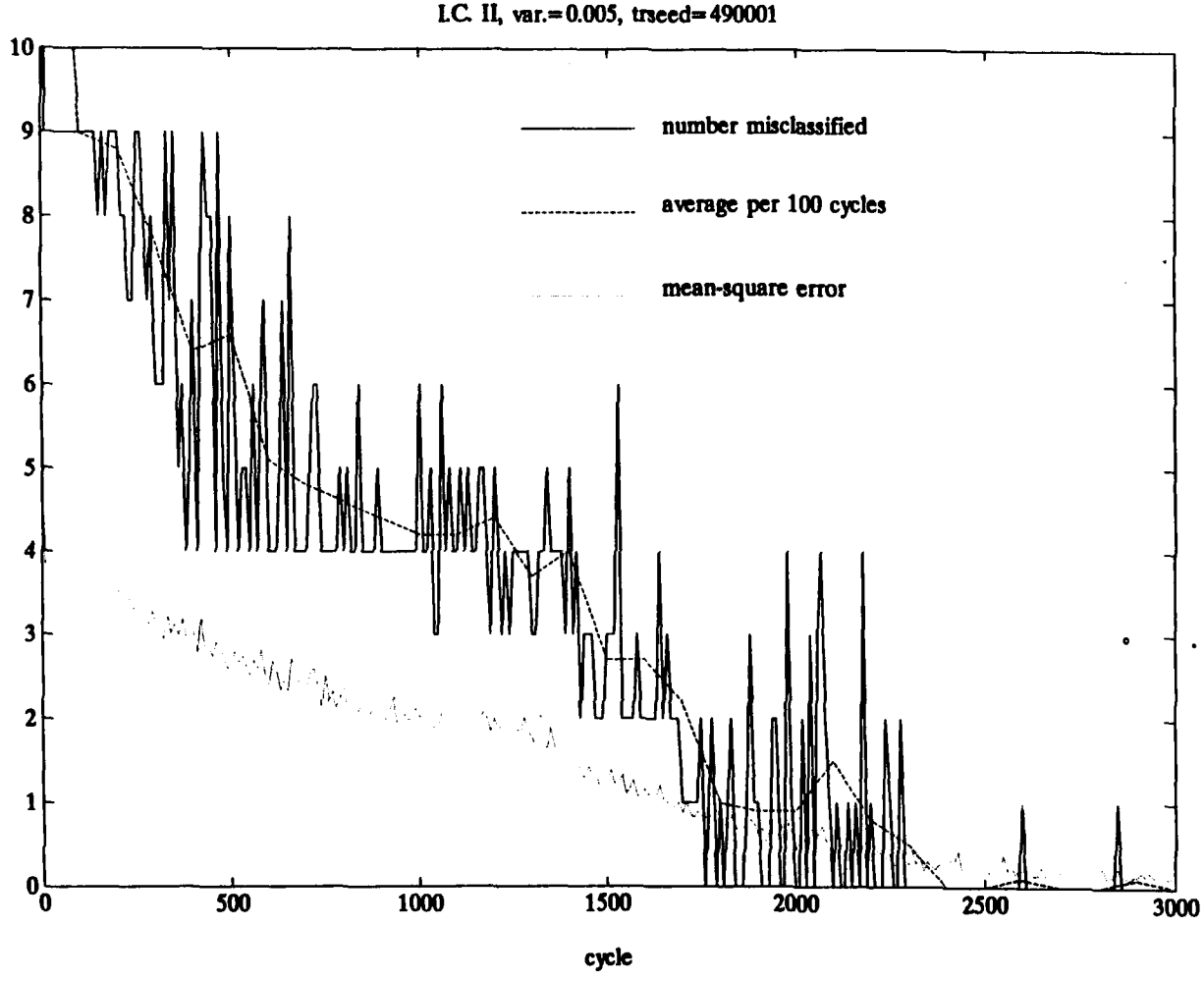


Figure 11. A typical learning curve corrupted by temporal noise, in this case in the SLM electronics. The averaging method allows approximation of convergence time by eliminating transients that occur after convergence.

We have found by examination of graphs of *noff* that this empirical method works well, in most cases. In some cases, the spikes are particularly large, and so 0.3 must be used in place of 0.2. Figure 11 shows the averages with respect to epoch. In those cases where this yields 3000, the simulation time was too short for convergence to occur (assuming it would have). In these instances, the next best measure is the average *noff* over the last 100 epochs (ten values):

```
mean(x.*(noff(292:301))
```

Initially, we considered each of the various imperfections by itself. Then we began combining key effects that seemed likely to be the most lethal combinations. Finally, we looked at the presence of all imperfections simultaneously.

1. The SLMs

Figure 12 shows a summary of the SLM imperfections. In this section we will expand upon the meanings of these imperfections and describe their effects on convergence.

a. The Effect of Malus's Law

The input and hidden vectors, as well as the weights, biases, and error terms δ_k and δ_j are all represented as the transmittances of SLMs. We have modeled them as Pockel's-effect devices, built around materials that exhibit a birefringence whose value is proportional to the applied electric field. The Pockel's cell has a known thickness and is placed between crossed polarizers. Polarized light passes through the cell and, as the voltage increases from zero, the output polarization gradually becomes elliptical, then circular, then elliptical with the major axis orthogonal to the input polarization, and finally linear orthogonal. Further increases in voltage, though avoided, begin to reverse this trend.

The analyzer is orthogonal to the input polarizer, so that with no voltage applied, the orthogonal polarization component is minimal and the transmittance is zero. In operation, then, the modulator's transmittance, what we call the optical weight *oW*, is given by

$$oW = \sin^2((\pi/2)eW),$$

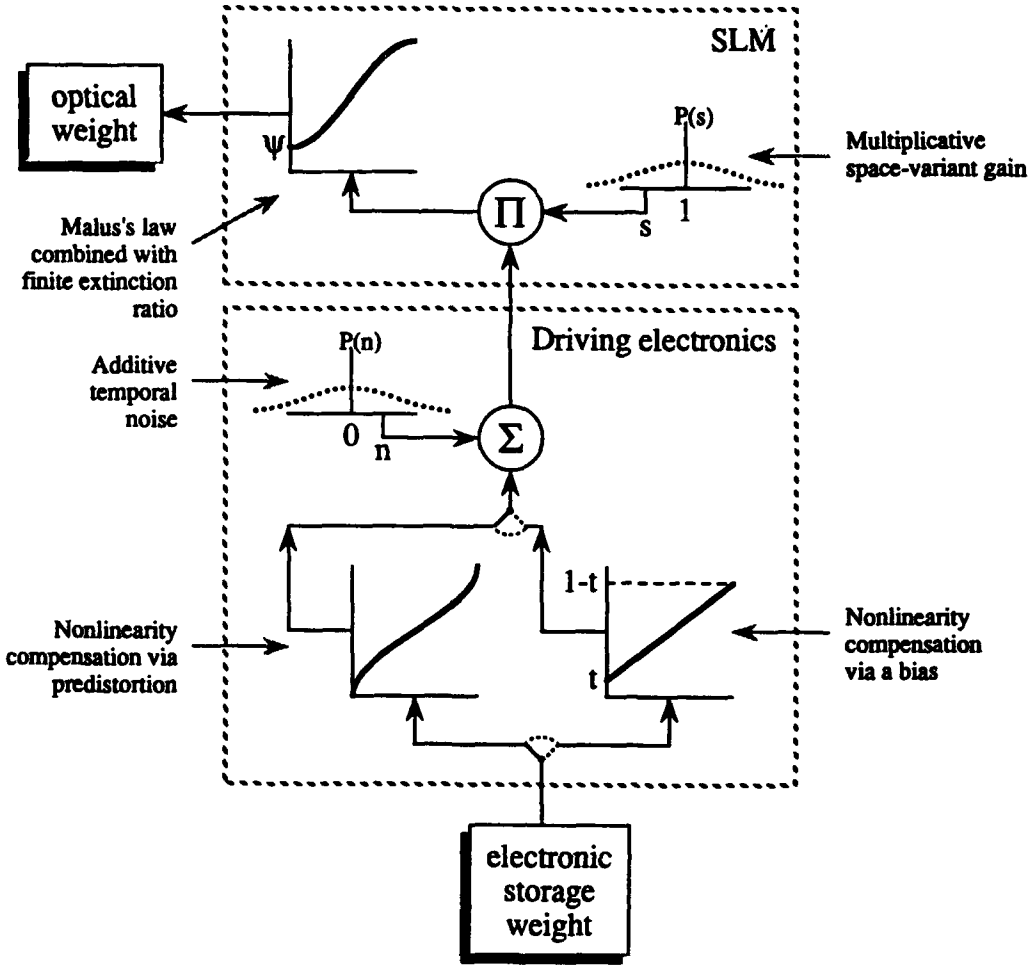


Figure 12. A model of the SLM element and its driving electronics.

where eW is the electrical value of the weight, with $0 \leq eW \leq 1$. (The value eW can be thought of as being propagated within the "driving electronics" box in Figure 12.) This relationship is an expression of Malus's law, and it immediately raises an important issue: since the transmittance is nonlinearly related to the applied voltage, is it necessary or desirable to compensate for this effect using electronic predistortion?

Let esW be the value in the electronic storage register where a weight or input is stored ("electronic storage weight" in Figure 12). The predistortion is expressed:

$$eW = (2/\pi)\sin^{-1}(esW^{\frac{1}{2}}).$$

This is shown in Figure 12, near the bottom left. We hold that while introducing such electronic predistortion into the 1-D SLMs is certainly feasible, having it in the 2-D SLMs may require undue electronic complexity. So, one set of our simulations is devoted to determining the effects of a nonlinear mapping between *esW* and *oW*.

Figure 13 shows the result of this effect. A sign-encoded pair of 2-D SLM elements passes light onto the associated detector elements. In each SLM element, the applied voltage is proportional to the magnitude. The amount of light detected, then, is proportional to the square of the sine of the magnitude, as shown in the curves. When (+) and (-) subelements are summed, the resulting transfer function exhibits three relatively flat regions: one near zero weight and two at the extreme (+) and (-) weight values.

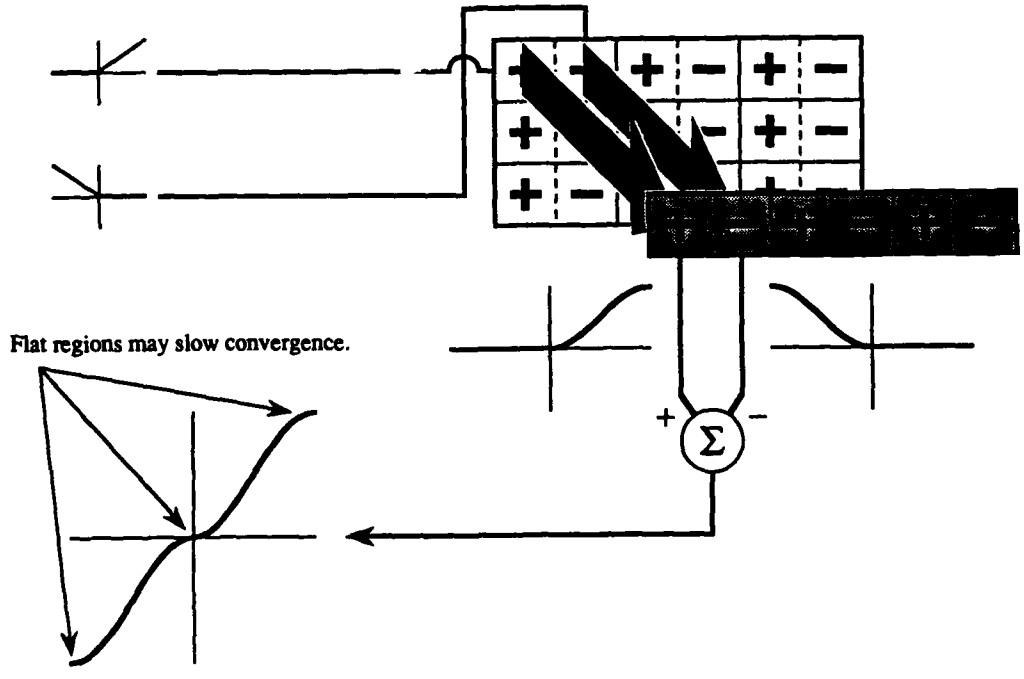


Figure 13. The effect of subelement sign encoding combined with the sine-squared nonlinearity.

Figure 14 shows a method to smooth out this transfer function. Instead of predistortion, simple biasing electronics reduce the range of input voltages, to avoid generation of subelement outputs in the flat regions. The price paid is, of course, a reduction in dynamic range, that is, an increase in sensitivity to stray voltages. Note

that this reduction has no apparent cost until noise in the driving electronics is included. If the normalized bias is t (tp2 in the Matlab simulations) the electronic register holds

$$eW = esW(1-2t) + t.$$

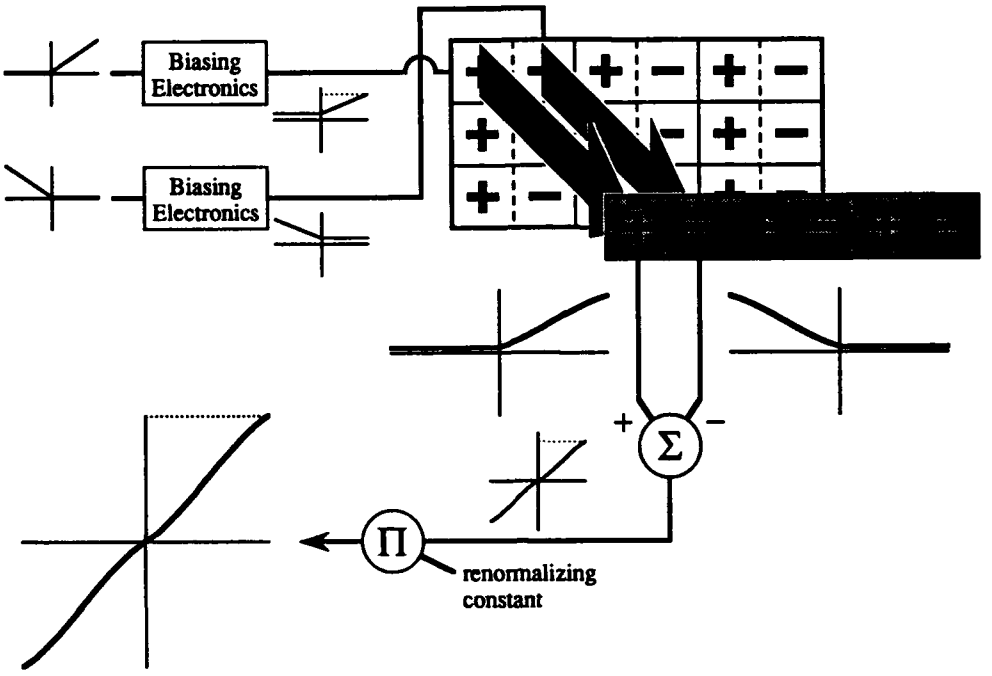


Figure 14. Reducing the transfer function nonlinearity by use of a bias.

This is shown graphically in Figure 12, near the bottom right. As shown in Figure 14, a renormalizing constant is necessary in the post-detection electronics. This effects a stretch in the vertical direction of the transfer function, preserving the integrity of the algorithm. The renormalizing constant is just

$$1/[\sin^2((\pi/2)(1-t))-\sin^2((\pi/2)t)].$$

Recall the variable a in Figures 5 and 6; it is the product of the renormalizing constant and h. Note that if $t = 0$ (no bias is used) or if predistortion is used, $a = h$.

In our simulations, we examined the effect of the nonlinearity without the bias and with different bias values. Figure 15 gives an idea of how much of a bias t

produces significant smoothing; for $t = 0.15$, the transfer function appears practically linear.

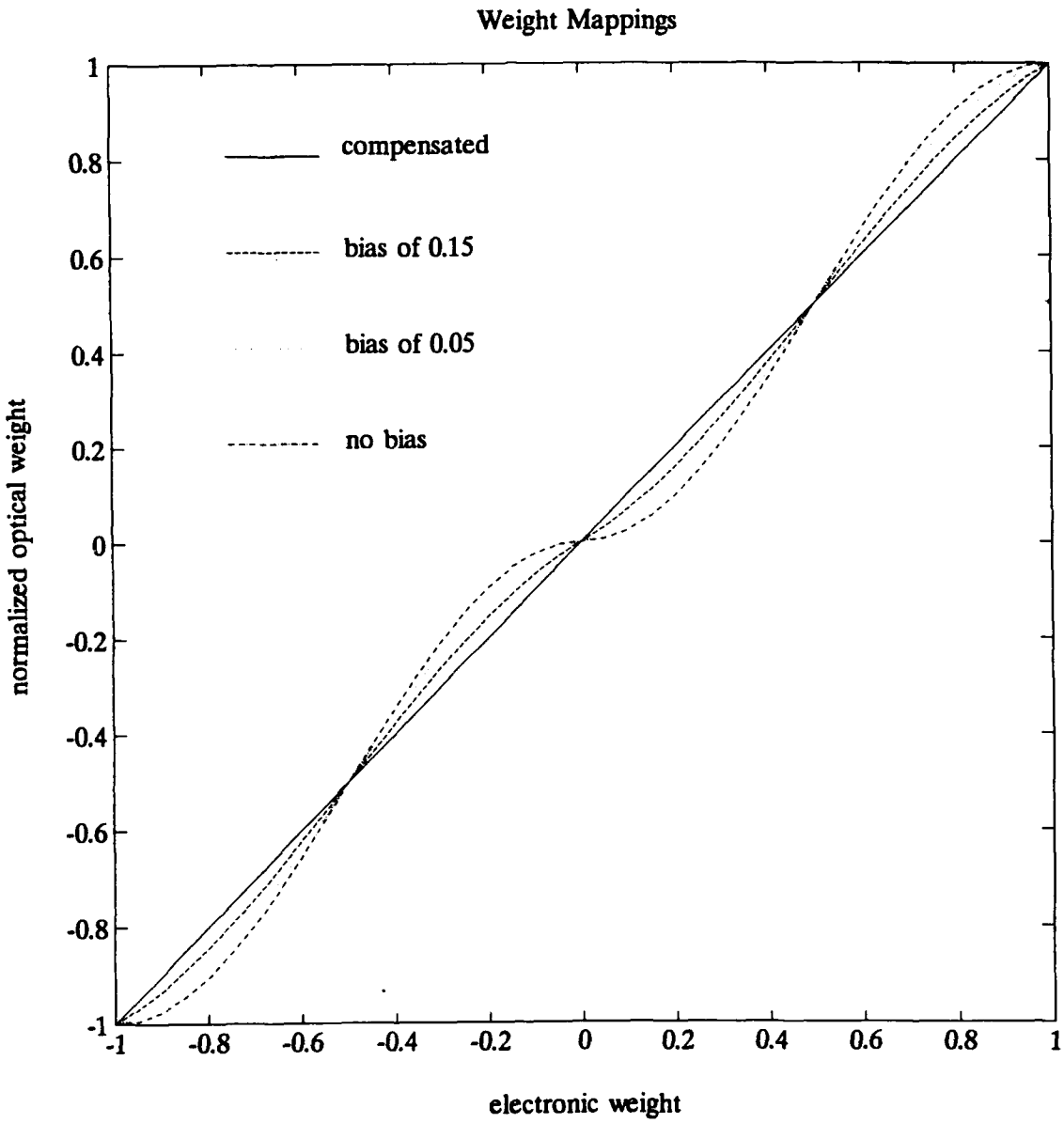


Figure 15. The SLM nonlinear transfer function for several biases.

Recall that in the 1-D SLMs, we have opted to retain predistortion compensation for the sine-squared nonlinearity.

For the four sets of initial conditions, we found that the nonlinearity has a detrimental effect on convergence time; nevertheless the system does eventually converge, sometimes. Introducing a bias significantly reduces the detrimental effect. Our biases are in the normalized regime, where a weight element can be at most one. Generally, as the bias approaches its limit of 0.5, the time to convergence approaches what it was in the case of predistortion compensation. Table I shows the time to convergence for each case.

TABLE I
Simulation results with and without compensation for Malus's law.

	time to convergence										
	compensated via bias										pure
I.C.†	zero	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	BEP
I	1870	1390	1240	1160	1100	1070	1040	1030	1020	1010	1010
II	[stuck]	1740	1530	1520	1580	1760	1750	1600	1420	1390	1400
III	>3000	2130	1800	1720	1700	1710	1730	1760	1780	1800	1810
IV	2300	1780	1720	1800	1850	1890	1930	1950	1960	1950	1950

†icseed values for I, II, III, and IV are 419043, 853252, 628191, and 107470, respectively.

b. Temporal Noise

The analog electronics driving the SLMs may exhibit temporal noise. This is modelled by additive Gaussian noise as shown in Figure 12, and is expressed in the equation

$$eW \Leftarrow eW + n$$

where n is a random number with a normal probability distribution having $\mu = 0$ and a specific variance σ^2.

It is important to note that this noise is added to the applied voltage rather than to the weight or input itself, whether or not it is preceded by predistortion; that is, it is applied to eW. Since the sine-squared nonlinearity is then applied to eW, a given noise

spike will have a greater effect for a weight or element near the middle (eW = 0.5) than at the range edges (eW = 0 or 1).

As we might expect, temporal noise gives a learning curve of mean-square error a spiked, jagged aspect. Figure 16 shows four learning curves, one from a noise-free trial, and three with different noise variances, but sharing the same random number generator seed. The effect of variance is to govern the size of the spikes as well as the degree to which the mean-square error approaches zero.

Table II shows the simulation results. The times to convergence are computed using the averaging metric referred to above. To ensure that our results are not too much an artifact of any one noise history, three different random number generator seeds were used.

TABLE II

Simulation results with temporal noise in the SLM driving electonics. Note that, except for the case of 0 noise, values are rounded to the nearest 100 by our averaging algorithm.

		time to convergence, $(noff]_{n=2900:10:3000}$					
			noise variance				
I.C.	training seed†	BEP	0.00125	0.0025	0.005	0.0087	0.015
I	a	1010	1100	1500	1600	2800	(0.7)
	b		1100	1300	2000	(0.3)	(1.4)
	c		1100	1400	1800	2800	(0.6)
II	a	1400	1700	1900	2100	2800	(0.8)
	b		1700	1900	2500	2900	(2.8)
	c		1700	2100	2300	2800	(1.3)
III	a	1810	1900	2000	2400	2900	(4.8)
	b		2000	2000	2400	(0.4)	(5)
	c		1900	2100	2300	2900	(4.5)
IV	a	1950	2400	2700	(0.3)	(2.5)	(3.5)
	b		1500	2200	2100	2900	(1.6)
	c		2400	1700	2400	(0.8)	(1)

†trseed values for a, b, and c are 893820, 205464, and 490001, respectively.

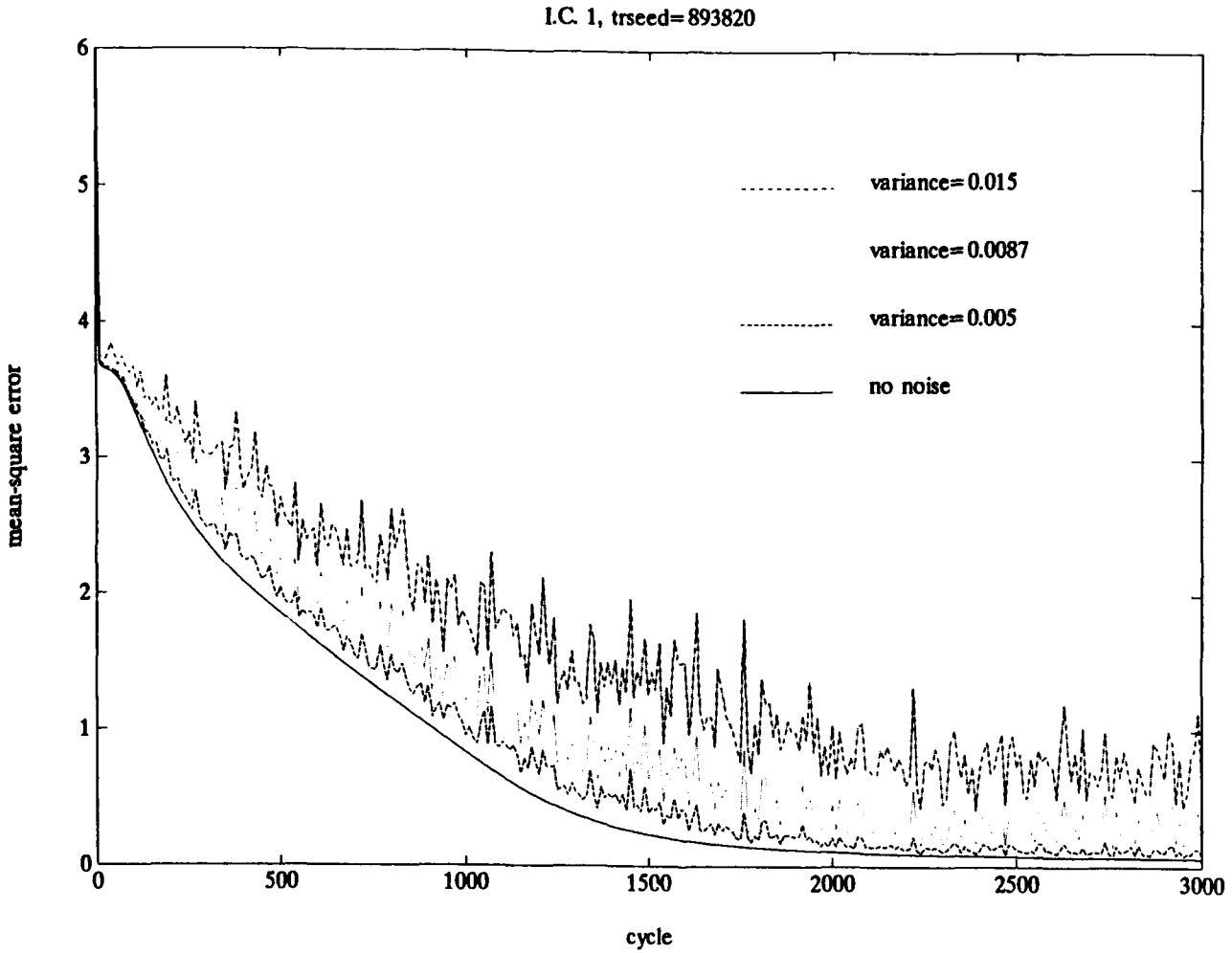


Figure 16. Learning curves for simulations with temporal noise in the SLM driving electronics.

c. Space-variant Gain

A given SLM may not be completely uniform across all the elements. This nonuniformity is most likely manifest as variation in thickness or electro-optic response. We model this as a set of element-dependent multipliers to the applied voltages. This set consists of random numbers s whose mean is one and which are normally distributed. That is,

$$eW \Leftarrow eW \times s.$$

Figure 12 illustrates this. Note that if a multiplier exceeds one, then the SLM element may induce a birefringent phase shift greater than $\pi/2$, creating a decrease in the weight. Therefore, with this nonuniformity, the transfer function for a particular element may not be monotonic.

An element-by-element nonuniformity in the device electro-optic response can be modelled by establishing a different random number multiplier for each element. The set of random numbers follows a Gaussian distribution with a mean of one and a known variance.

With reference to Figure 7, the SLMs encode o_i, DW_{ji}, o_j, DW_{ji}, δ_k (unipolar), δ_k (bipolar), and δ_j. For our problem, with $I = 2$, $J = 4$, and $K = 1$, the architecture then possesses only fifty-three elements. In the regime of statistics, this is not a large number. The point is, it is particularly easy to produce a "bad seed" phenomenon, in the simulations. That is, though the variance of the multipliers may be small, the more critical elements happen to have the worst deviations—or vice versa. To allow for this possibility, we performed these simulations using four different seeds for the random number generator which produces the multipliers.

Table III shows the results. Judging by the low variances, it takes only a small degree of electro-optic nonuniformity to render the architecture unfit for use. While it may appear that in many instances, convergence time is being reduced, the point is that it is being drastically changed, and that a different problem of the same size may very well get stuck.

TABLE III
Simulation results with nonuniformity in the SLM electro-optic response.

I.C.	seed†	time to convergence: BEP	variance of element response multipliers: 0.003125	0.00625	0.0125	0.025	0.05
I	a	1010	1000	990	990	[stuck]	[stuck]
	b		1010	1020	1030	1110	1450
	c		1020	1030	1050	1280	[stuck]
	d		1010	1020	1030	1090	2490
II	a	1400	1490	2000	2100	2420	[stuck]
	b		1720	1420	1380	1440	1690
	c		1460	1630	1870	1790	2980
	d		1560	1810	1390	1870	1780
III	a	1810	1810	1940	2850	[stuck]	[stuck]
	b		1920	2090	>3000	[stuck]	[stuck]
	c		1750	1830	>3000	>3000	[stuck]
	d		1810	1850	>3000	[stuck]	1770
IV	a	1950	1270	1530	1940	2020	[stuck]
	b		1940	1250	1620	1590	[stuck]
	c		1660	1730	2020	2400	[stuck]
	d		2540	2380	2520	>3000	[stuck]

†svseed values for a, b, c, and d are 598392, 774772, 200493, and 920145, respectively.

d. Finite Extinction Ratio

The modulator devices are placed between crossed polarizers, themselves a possible source of error. The error occurs when the polarizers' extinction ratio is finite, as depicted in Figure 12. If ψ is the reciprocal of the extinction ratio, the modulator's transmittance is given by

$$oW = (1-\psi)\sin^2((\pi/2)eW) + \psi.$$

In our design, all elements of a given SLM share the same polarizer and analyzer. Regardless of which has the finite extinction ratio, the result should be pretty much the same. For simplicity, we assumed that the same quality of polarizer is used throughout, and subject all operations to an overall extinction ratio. For example, for an extinction ratio of 100, all SLM elements set to zero really pass 0.01; those set to one pass one, and the entire curve is adjusted for the intermediate values.

Initially we tried a broad range of extinction ratio values, using just I.C. II. Table IV shows these preliminary results. Then, using all four sets of I.C.s, we narrowed our range of interest to that shown in Table V.

TABLE IV
Preliminary results with finite extinction ratio in the SLMs, using I.C. II.

extinction ratio	time to convergence
∞ (pure BEP)	1400
2154	1400
1000	1400
464.2	1400
215.4	1420
100	1480
46.42	1660
25.14	1800
10	2040

TABLE V
More detailed results with finite extinction ratio in the SLMs.

	time to convergence								
	extinction ratio								
I.C.	∞	100	56.23	31.62	17.18	10	5.623	3.162	1.778
I	1010	1060	1110	1190	1360	1730	2860	>3000	>3000†
II	1400	1480	1590	2090	1830	2040	2990	>3000	>3000†
III	1810	1830	1860	1920	2040	2360	>3000	>3000	>3000†
IV	1950	1990	2030	2090	2260	2230	>3000	>3000	>3000†

†It is uncertain whether these trials were on their way to convergence.

Apparently, the system is very tolerant of finite extinction ratio, showing convergence even for values less than 10. For the most part, the poorer extinction ratio seems simply to decrease the effective learning rate.

2. The Optical Imaging System

The optical imaging system consists of that set of lenses and polarizing beamsplitters which image one device plane onto another. In a given operation, such as a vector-matrix multiplication, two optical imaging systems are employed: one imaging the inputs onto the weight matrix, the other imaging the weight matrix onto a detector array. The chief mechanism by which optical imaging systems introduce error is crosstalk.

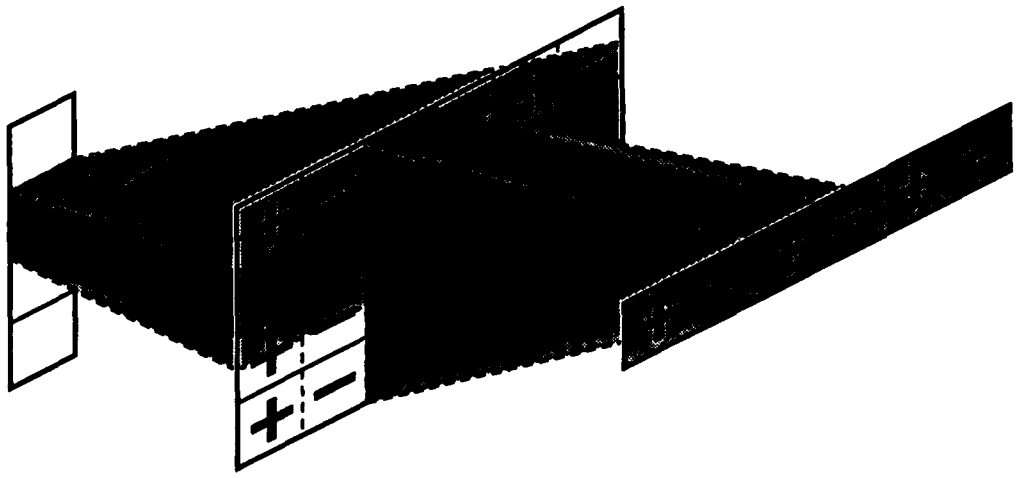


Figure 17. Crosstalk in an optical vector-matrix-multiplier.

The process of crosstalk is illustrated in Figure 17. Each input element broadcasts to a weight SLM row. Some of the light, say 90%, reaches the appropriate row. The other 10% is divided between the two adjacent rows. Similarly, each column is optically summed onto one detector element. 90% of the light reaches that element; the other 10% goes to the elements adjacent to it. Note that this means some of the light designated for $o_j(+)$ adds to $o_j(-)$ and $o_{j-1}(+)$.

When we first proposed this architecture, we considered two encoding layouts. In one, each individual weight is divided into two side-by-side subelements, namely the (+) and (-) subelements. This is the approach shown in Figure 17. Another layout would consist of two noninterlaced submatrices, namely the (+) submatrix and the (-) one. We embraced the former approach based on a series of trial vector-matrix multplies using both approaches. We found the former apprach, that of side-by-side subelements, to be more crosstalk tolerant in the final answer.

As can be seen in Table VI, the alorithm is sensitive to even small amounts of crosstalk. Differences in convergence time result from as little as a tenth of a percent crosstalk. Curiously, the convergence time may decrease. At 1% crosstalk, convergence times are significantly changed, but the algorithm does converge. At 6%, the network performance is severely corrupted.

TABLE VI
The effects of crosstalk upon convergence.

	time to convergence, $(noff)_{n=2900:10:3000)}$										
	crosstalk										
I.C.	0%	0.05%	0.1%	0.2%	0.4%	0.6%	1%	1.4%	2.8%	6%	12%
I	1010	1020	1030	1040	1080	1110	1180	1260	1620	(4)	[stuck]
II	1400	1400	1410	1440	1500	1580	1820	2350	2120	(4)	[stuck]
III	1810	1810	1820	1830	1850	1870	1940	2030	2380	2360	(6)
IV	1950	1940	1930	1910	1870	1830	1990	1820	1880	(3)	(10)

Figure 18 depicts some examples of learning curves for a system with crosstalk. It is interesting to note that, for this particular initial condition set (as well as others), increased crosstalk appears to shift the curves to the right. This suggests that crosstalk has its greatest effect in the beginning stages of learning.

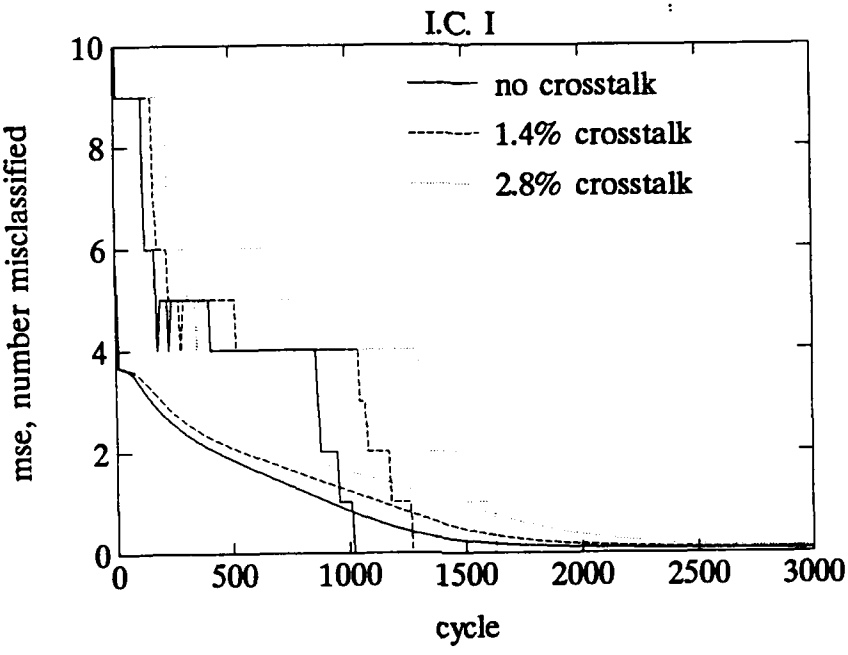


Figure 18. Learning curves for simulations with crosstalk.

3. The Detectors

We have incorporated into the architecture two varieties of detector arrays. The three linear arrays that compute o_j, o_k, and $\sum \delta_k W_{kj}$ are PIN detector arrays. The two 2-D arrays are CCD arrays. They accumulate charge over all passes in an epoch and then impart this charge to the 2-D SLMs after electronic amplification and other processing.

Both types of detector exhibit shot noise and thermal noise. Shot noise is signal-dependent; thermal, signal-independent. The derivations below concern the standard deviations of each type of noise in each type of detector. They show how to express these values in terms of the algorithmic units and as a function of maximum signal-to-noise ratios. This allows one to simply specify these ratios rather than the physical constants (e.g. operating temperature) and then run the simulations.

a. PIN Shot Noise

The shot noise manifests itself as variations in the light-induced photocurrent. The standard deviation is

$$\sigma_{shot} = (2qPSB)^{\frac{1}{2}}$$

in Amperes, where q is the electronic charge, P is the incident power in Watts, S is the sensitivity, or responsivity, in Amperes/Watt, and B is the bandwidth in Hz. Consider the first-layer forward pass. Let P_{max} denote the power when all SLM elements are transparent (have a value of 1). Let $oNet_j$ be the normalized value assigned to *one* of the two subelements for a given Net_j value. Since $oNet_j$ can be at most $I + 1$ (the "1" is for θ_j), we can recast the standard deviation as

$$\sigma_{shot} = \{2q[oNet_j/(I + 1)]P_{max}SB\}^{\frac{1}{2}}.$$

The signal-to-noise ratio (SNR) corresponding to maximum available detected intensity is

$$SNR_{max} = P_{max}S/(2qP_{max}SB)^{\frac{1}{2}}.$$

Using this equation, we can substitute for $P_{max}S$ in the shot noise equation to yield

$$\sigma_{shot} = P_{max}S[oNet_j/(I + 1)]^{\frac{1}{2}}/SNR_{max}$$

in Amperes. In terms of $oNet_j$,

$$\sigma_{shot}(oNet_j) = \sigma_{shot}[(I + 1)/(P_{max}S)]$$

$$= [oNet_j(I + 1)]^{\frac{1}{2}}/SNR_{max}.$$

Note that when $oNet_j$ reaches its maximum, $I + 1$, the shot noise σ becomes $(I + 1)/SNR_{max}$.

For our simulations, we chose a broad range of values for the SNR_{max}. We also saw the need for only one training seed, unlike the in other temporal noise simulations. This is because many more random numbers are called in the detection equation than in the weight updates, so the "bad seed" phenomenon is less likely to occur.

The results of the simulation are shown in Table VII. The SNR_{max} threshold below which significant changes in convergence time occurs seems to be around 21. The value below which convergence is prevented completely is not far from this.

TABLE VII

Shot noise in the PIN detectors. Note that, except for the case of ∞, values are rounded to the nearest 100 by our de-spiking algorithm.

	time to convergence, $(noff]_{n=2900:10:3000}$						
	SNR_{max}						
I.C.	∞	215.4	100	46.42	21.54	10	4.462
I	1010	1000	1000	1000	1100	(0.3)	(5.4)
II	1400	1400	1400	1500	1700	(0.5)	(6.3)
III	1810	1800	1800	1800	1900	(0.5)	(8.1)
IV	1950	2100	2100	2100	2500	(1.2)	(7.6)

b. PIN Thermal Noise

The thermal noise is a signal-independent function of such parameters as temperature:

$$\sigma_{thermal} = (4k_B TB/R_L)^{\frac{1}{2}}$$

in Amperes, where k_B is Boltzmann's constant, T is the temperature in Kelvins, B is the bandwidth, and R_L is the load resistance of the detector electronics, in Ohms.

The SNR corresponding to maximum available detected intensity is

$$SNR_{max} = P_{max}S/(4k_B TB/R_L)^{\frac{1}{2}}.$$

Substitution for $R_L^{\frac{1}{2}}$ in the thermal noise equation yields

$$\sigma_{thermal} = P_{max}S/SNR_{max}$$

in Amperes. In terms of $oNet_j$,

$$\sigma_{\text{thermal}}(oNet_j) = \sigma_{\text{thermal}}((I + 1)/(P_{\max}S))$$

$$= (I + 1)/\text{SNR}_{\max}.$$

The simulation results, given in Table VIII, point to a somewhat greater sensitivity to $\text{SNR}_{\max}$ than was seen for shot noise. This is not surprising; note that the standard deviation for thermal noise is signal-independent, actually corresponding to that shot noise whose signal is fixed to the greatest value, $I + 1$.

TABLE VIII
Thermal noise in the PIN detectors.

	time to convergence, $(\text{noff})_{n=2900:10:3000}$						
	$\text{SNR}_{\max}$						
I.C.	∞	215.4	100	46.42	21.54	10	4.462
I	1010	1000	1000	1400	(0.8)	(9.1)	(8.8)
II	1400	1400	1500	2000	(0.9)	(9.0)	(8.8)
III	1810	1800	1800	2100	(5.3)	(8.9)	(8.8)
IV	1950	2100	2800	2100	(4.3)	(9.1)	(8.7)

c. CCD Shot Noise

The CCD noise equation derivations proceed similarly, except that we measure quantities in photoelectrons accumulated, rather than current. The CCD noise calculations are used once at the end of each epoch.

The standard deviation of the CCD shot noise is

$$\sigma_{\text{shot}} = CN^{\frac{1}{2}}$$

in photoelectrons actually released, where C is a constant, and N is the number of photons absorbed (assuming unity quantum efficiency). Consider the first-layer outer product. Let $N_{\max}$ denote the number of photons that the detector element would absorb in all R passes in an epoch were the SLM elements transparent (valued at 1). Let oDW_{ji} be the normalized value assigned to *one* of the two subelements for a given DW_{ji} value. The most oDW_{ji} can be, then, is R. We then recast the standard deviation

$$\sigma_{\text{shot}} = C[(oDW_{ji}/R)N_{\max}]^{\frac{1}{2}}.$$

The SNR corresponding to maximum available absorbed photons is

$$SNR_{max} = N_{max}/[C(N_{max}^{\frac{1}{2}})]$$

Substitution for N_{max} in the shot noise equation yields

$$\sigma_{shot} = N_{max}(oDW_{ji}/R)^{\frac{1}{2}}/SNR_{max}$$

in photoelectrons. In terms of oDW_{ji},

$$\sigma_{shot}(oDW_{ji}) = \sigma_{shot}(R/N_{max})$$

$$= (oDW_{ji}R)^{\frac{1}{2}}/SNR_{max}.$$

The simulation results are given in Table IX. The transition from the noise just making a difference to the point at which convergence is prevented is more gradual than it was for the PIN shot noise case. The algorithm thus seems more tolerant of CCD shot noise than PIN shot noise.

TABLE IX
Shot noise in the CCD detectors.

	time to convergence, $(noff]_{n=2900:10:3000)}$						
	SNR_{max}						
I.C.	∞	215.4	100	46.42	21.54	10	4.462
I	1010	1000	1000	1000	1100	1200	(5.0)
II	1400	1400	1400	1500	1700	2200	(5.6)
III	1810	1900	1900	1900	2200	1200	(4.8)
IV	1950	2000	2500	1800	1900	2000	(5.0)

d. CCD Thermal Noise

The CCD thermal noise is independent of the number of incident photons, and is instead a function of an arbitrary number n of photons:

$$\sigma_{thermal} = Cn^{\frac{1}{2}}$$

in photoelectrons actually released, where C is a constant.

The SNR corresponding to maximum available absorbed photons is

$$\mathrm{SNR}_{\max} = N_{\max}/[C(n^{\frac{1}{2}})]$$

Substitution for C in the thermal noise equation yields

$$\sigma_{\mathrm{thermal}} = N_{\max}/\mathrm{SNR}_{\max}$$

in photoelectrons. In terms of oDW_{ji},

$$\sigma_{\mathrm{thermal}}(oDW_{ji}) = (N_{\max}/\mathrm{SNR}_{\max})(R/N_{\max})$$

$$= R/\mathrm{SNR}_{\max}.$$

As shown in Table X, the algorithm is quite sensitive to thermal noise in the CCDs. Generally, convergence is impeded for $\mathrm{SNR}_{\max}$ values less than 100 or so. Note that for the three lowest $\mathrm{SNR}_{\max}$ values, the number misclassified is the same for all four sets of initial conditions. Indeed, at these values, the learning curves are practically identical and characterized chiefly by the noise.

TABLE X
Thermal noise in the CCD detectors.

	time to convergence, $(\mathrm{noff}]_{n=2900:10:3000}$						
	$\mathrm{SNR}_{\max}$						
I.C.	∞	215.4	100	46.42	21.54	10	4.462
I	1010	1100	1800	(9.8)	(12.2)	(13.0)	(12.5)
II	1400	1900	(4.0)	(10.2)	(12.2)	(13.0)	(12.5)
III	1810	1500	1900	(9.8)	(12.2)	(13.0)	(12.5)
IV	1950	1900	(2.3)	(10.0)	(12.2)	(13.0)	(12.5)

e. Realistic Detection Parameters

For most of the imperfections, crosstalk for example, it is not difficult to suggest realistic expectations of what a given system might be capable of. At this point, we shall briefly discuss the PIN detectors.

Recall the PIN shot noise derivation: along the way we found

$$\mathrm{SNR}_{max} = P_{max}S/(2qP_{max}SB)^{\frac{1}{2}}.$$

The electronic charge $q = 1.6\times10^{-19}$ Coulombs. Let the bandwidth $B = 10^7$/s and the responsivity $S = 1$ Ampere/Watt. For each detector element, assume the maximum light power availability to be $P_{max} = 10^{-5}$ Watts. This gives $\mathrm{SNR}_{max}(\mathrm{shot}) \simeq 1766$.

In the PIN thermal noise derivation, we found

$$\mathrm{SNR}_{max} = P_{max}S/(4k_BTB/R_L)^{\frac{1}{2}}$$

where $k_B = 1.38\times10^{-23}$ J/K, and we let $T = 300$K. R_L is the load resistance, which is typically 100Ω. $\mathrm{SNR}_{max}(\mathrm{thermal}) \simeq 246$.

C. Combinations of Key Effects

While two or more given effects may by themselves be small enough to little affect the network's convergence property, they may combine much more lethally—or cancel one another. Our approach for any combination is to use two sets of values. The "minimal" set corresponds to those values which by themselves had little effect. The "maximal" set are those values which by themselves were enough to delay convergence significantly, but not enough to altogether prevent it from occuring within the normal number of cycles.

1. Temporal Noise and Malus's Law in the Weights

As we discussed in Section IV. B. 1. a., Malus's Law effects a nonlinearity in the weight transfer function. Short of predistortion, this nonlinearity may be reduced using a bias, at the expense of dynamic range. We should expect the loss in dynamic range to appear as an increase in the effective temporal noise. (This increase is due to the renormalizing constant which is necessary to preserve the integrity of the algorithm.)

Furthermore, we wished to test the following hypothesis: that there should be a "sweet spot" in the bias for a given, low, temporal noise value. That is, we can find a bias which is large enough to help overcome the flat regions of the nonlinearity but small enough not to significantly increase the effective temporal noise.

To see this effect, we used the very small temporal noise variances of 0.005, 0.0025 and 0.00125. Table XI shows the results; Figure 19 shows them graphically. Indeed, we can observe the existence of a bias "sweet spot," whose value increases with decreasing SLM noise. That is, as SLM noise is reduced, larger biases can be used.

TABLE XI
The combining of the Malus's Law nonlinearity with SLM temporal noise.†

	time to convergence, $(noff]_{n=2900:10:3000)}$				
			bias		
variance	0	0.1	0.2	0.3	0.4
0.005	2000	2000	(0.3)	(1.2)	(4.3)
0.0025	2000	1500	1500	2100	(0.5)
0.00125	1900	1500	1400	1400	2100

†I.C. I, trseed=205464.

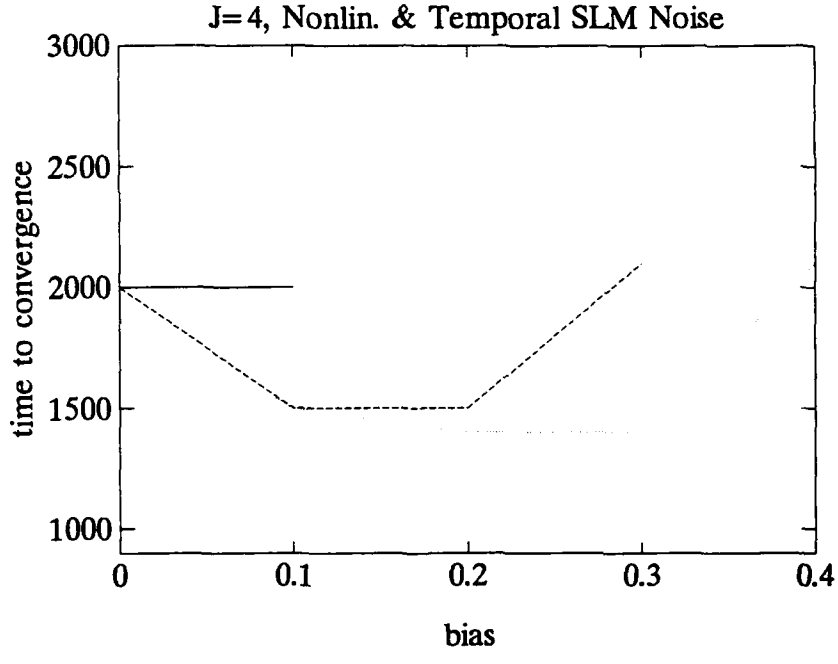


Figure 19. The existence of a "sweet spot" in bias, whereat the bias's benefit and harm are in balance. The solid curve represents a noise variance of 0.005; the dashed curve, 0.0025; the dotted curve, 0.00125.

2. Crosstalk and Shot Noise

As a comparison between Tables XII and VI shows, the inclusion of shot noise (in both the PIN and CCD detectors) compounded with the effect with crosstalk,

increases—usually—the convergence time. Table XXII shows simulation results for two sets of crosstalk and shot noise values. Note that we did not use the averaging method to obtain convergence time, even though temporal noise was present—the plots of *noff* appeared to be free of the spikes that had necessitated the method for SLM temporal noise data.

TABLE XII
The combining of crosstalk and shot noise.†

I.C.	time to convergence BEP	minimal‡	maximal††
I	1010	1040	1440
II	1400	1440	2480
III	1810	1820	1840
IV	1950	2120	2550

†trseed=893820
‡0.2% crosstalk, SNR_{max} = 100 for both PIN shot and CCD shot noise.
††1% crosstalk, SNR_{max} = 21.54.

3. The Complete Detector Noise Models

Generally, the effects of shot noise and thermal noise in all elements of detection do accumulate. However, as the data in Table XIII show, the different effects do not appear to form a lethal combination.

TABLE XIII
The complete detector noise models.†

I.C.	time to convergence BEP	minimal‡	maximal††
I	1010	1100	1900
II	1400	1600	2000
III	1810	2000	2500
IV	1950	1300	2400

†trseed=893820
‡SNR_{max}(PIN and CCD shot) = 100, SNR_{max}(PIN thermal) = 215.4, SNR_{max}(CCD thermal) = 464.2.
††SNR_{max}(PIN shot) = 21.54, SNR_{max}(PIN thermal) = 46.42, SNR_{max}(CCD shot) = 10, SNR_{max}(CCD thermal) = 215.4.

4. The Complete, Compensated SLM Noise Model

We have never ruled out the possibility of using electronic predistortion to compensate for the SLM nonlinearity. As we mentioned in Section IV. B. 1., it may merely introduce undue complexity into the 2-D SLMs. Recall, too, that in the 1-D SLMs, we always incorporate this predistortion.

Since the lack of a predistortion seems to magnify SLM noise sensitivity, we opted to test the complete SLM model with compensation. The results appear in Table XIV.

TABLE XIV
The complete, compensated SLM noise model.†

I.C.	BEP	time to convergence minimal‡	maximal††
I	1010	1100	2000
II	1400	1800	2200
III	1810	1900	2800
IV	1950	1600	2000

†trseed=205464; svseed=200493.
‡$\sigma^2 = 0.00125$, $\sigma^2(s) = 0.004$, ext. ratio = 215.4.
††$\sigma^2 = 0.005$, $\sigma^2(s) = 0.01$, ext. ratio = 31.62.

D. Hidden-layer Redundancy

When more hidden units than are needed to solve a problem are present, the back propagation algorithm often converges more quickly. For the 2-D corners problem, we have found that back propagation chooses four hidden units, emphasizes them, and diminishes the others, all via W_{kj}. Figure 20 dramatically illustrates this trend in learning.

It is possible, too, that an optical architecture with hidden-layer redundancy will exhibit greater immunity to certain types of noise, and perhaps increased sensitivity to others. In any case, increasing J in an optical architecture is certainly easy, and can be done at a substantially lower cost in computation time than for a similar increase on a serial electronic computer.

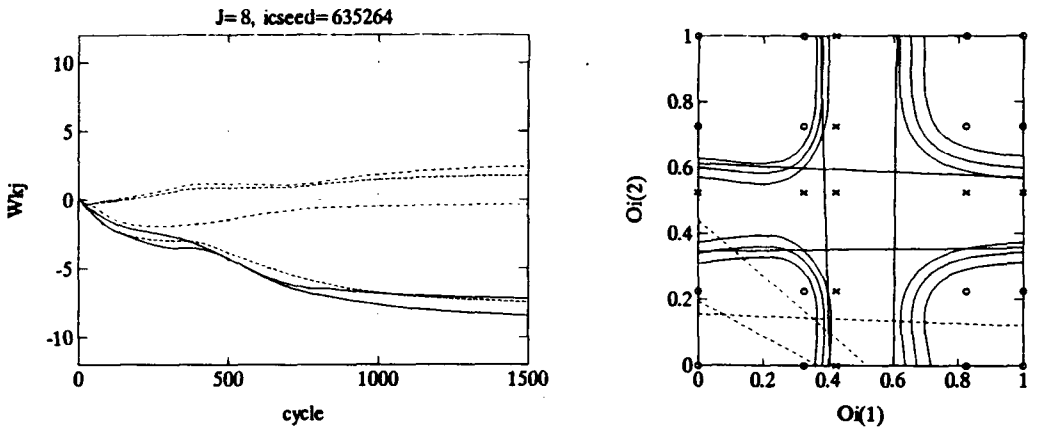


Figure 20. Evolution of connection strengths from the hidden layer of size $J = 8$, solving a problem requiring only $J = 4$. Also shown is a contour plot; the weakly connected hidden units produce the dashed decision lines (one outside of view).

To ascertain such changes in noise sensitivity, we performed a set of simulations using the same corners problem, but with $J = 8$—twice the necessary number of hidden units. Some preliminary runs using straight back propagation pointed us to the use of a new learning rate: $\eta = 1$. We still initialized the network using Gilbert's method, with a hill steepness of 4, as in the $J = 4$ simulations. Also, we chose four sets of initial conditions; the learning curves associated with these are shown in Figure 21. Note that, though the curves appear to vary, the convergence times are more similar than they were for $J = 4$. This is probably due to the fact that Gilbert's method makes the initial conditions more alike, and more so with redundant hidden units. The times to convergence are 720, 890, 960, and 920 cycles, respectively. We ran each of the $J = 8$ simulations until $N = 2500$.

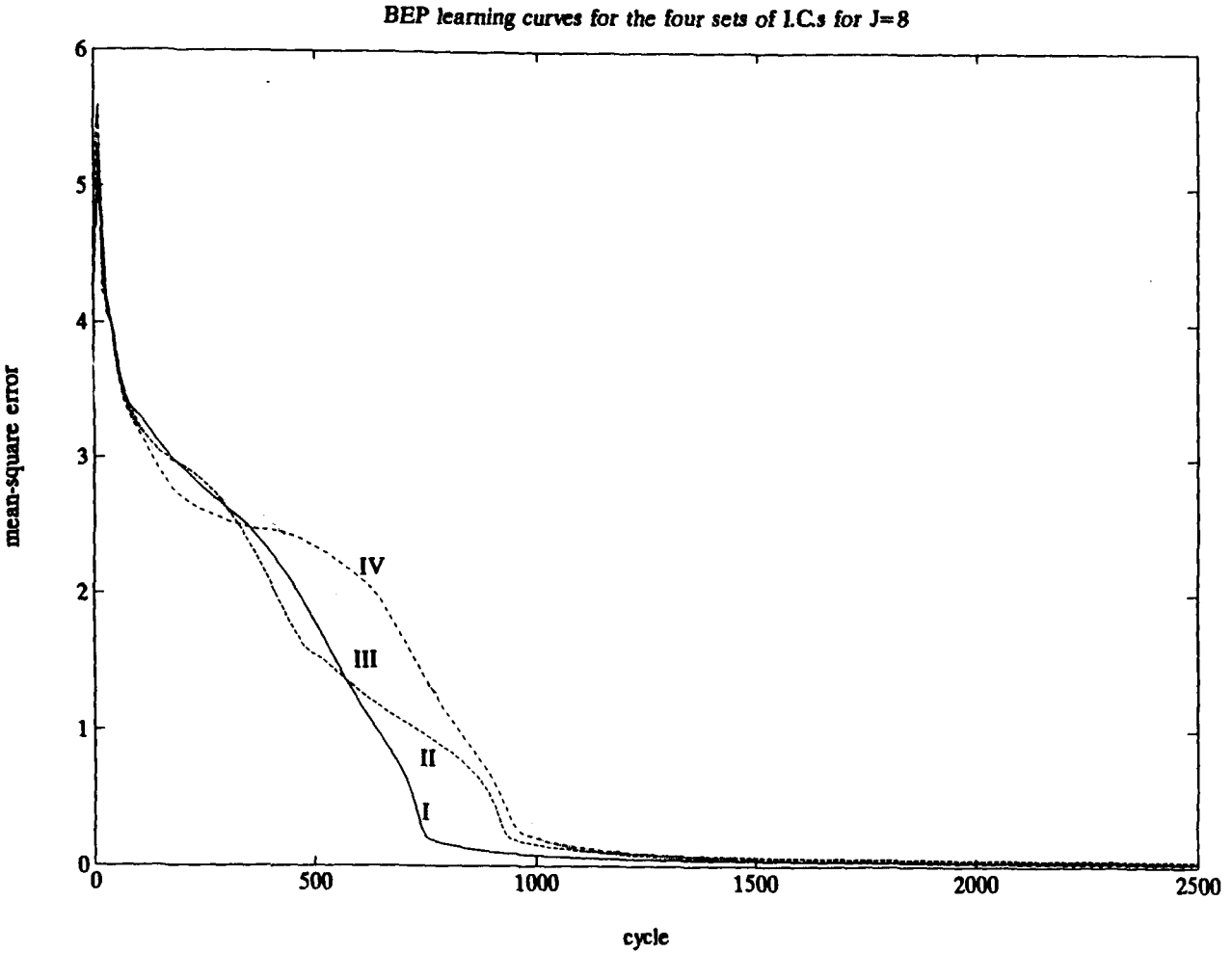


Figure 21. *Four representative learning curves generated by back propagation, with* $J = 8$, on the training data of Figure 7.

1. The SLMs

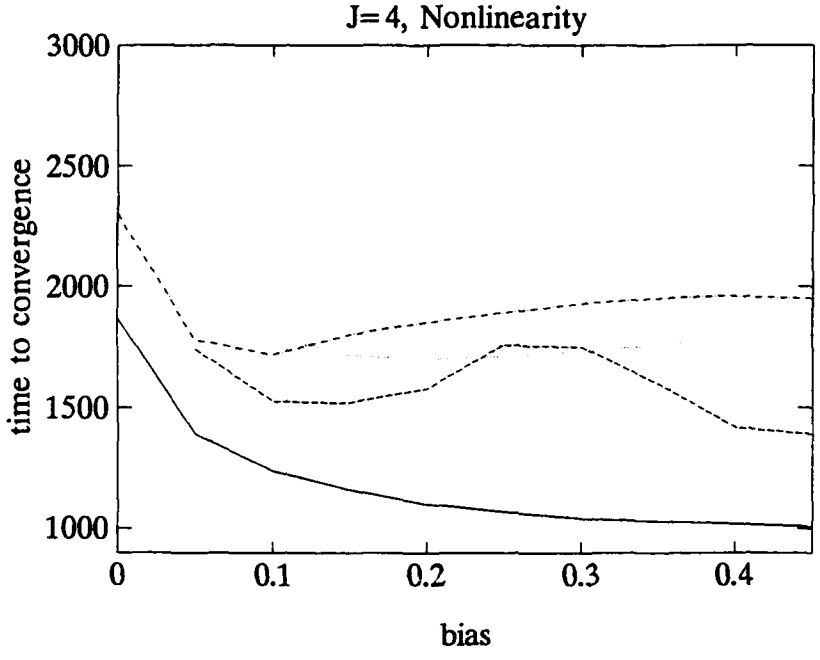
a. The Effect of Malus's Law

As we had when J was 4, we disabled predistortion compensation for the SLM nonlinearity, replacing it with a biased input (see Figure 12). For the most part, the absence of a bias resulted in the longest convergence times. But, as in the $J = 4$ case, as the bias approaches its limit, the convergence time approaches its value in the compensated case. Table XV shows the data. Figure 22 provides for rapid comparison of the convergence data for both J values.

TABLE XV
Simulation results with and without compensation for Malus's law, for $J = 8$.†

	time to convergence										
	compensated via bias										pure
I.C.†	zero	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	BEP
I	1610	1010	830	830	780	750	730	720	720	720	720
II	1400	1220	1240	1140	940	920	900	900	890	890	890
III	1570	1060	940	900	860	890	960	960	960	960	960
IV	[stuck]	1020	890	840	880	890	890	900	910	910	920

†icseed values for I, II, III, and IV are 822182, 332650, 237064, and 825089, respectively.



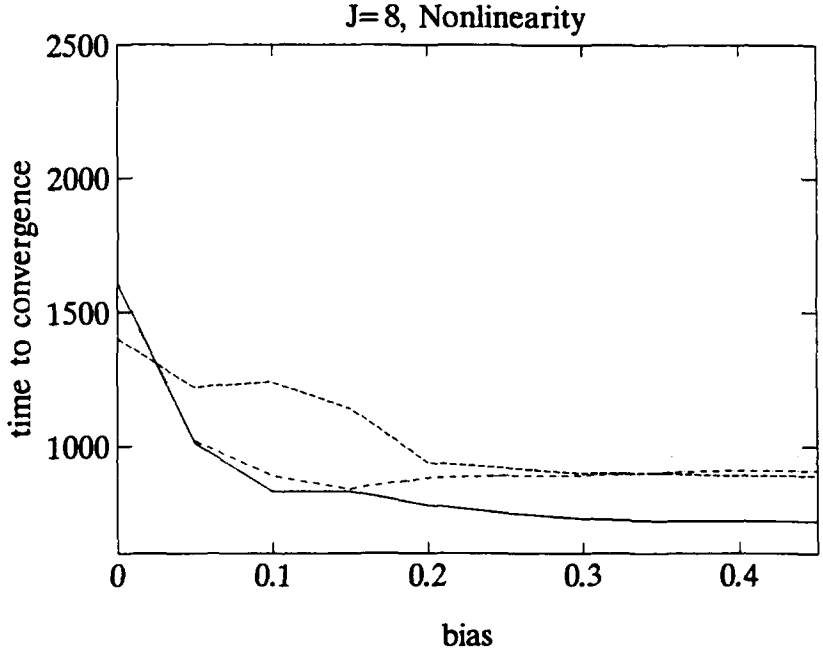


Figure 22. Convergence time vs. bias for two networks with different hidden layer sizes.

b. Temporal Noise

The simulations incorporating temporal noise generated learning curves whose intermittent spikes in *noff* were taller. Therefore we modified our averaging metric to suppress taller spikes:

```
xb=100:100:3000;
max(xb.*(mean(reshape(noff(2:301),10,30))>0.3))
```

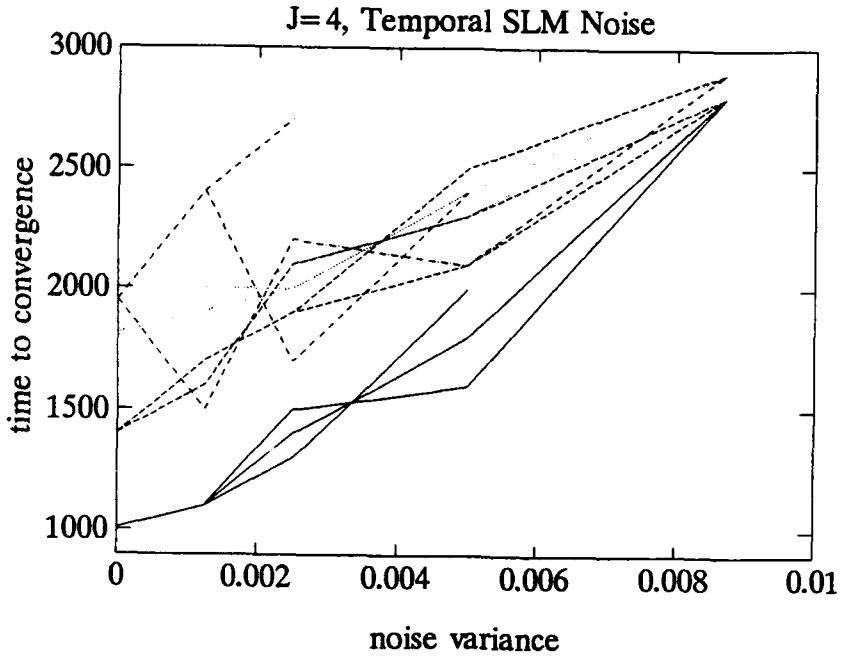
The convergence time data appear in Table XVI. Figure 23 shows graphically the results for both J values. The different line types correspond to different sets of initial conditions; note three lines per condition (for the three training seeds) appear. It is difficult to deduce any significant change in SLM noise sensitivity with J.

TABLE XVI

Simulation results with temporal noise in the SLM driving electonics, for $J = 8$. Note that, except for the case of 0 noise, values are rounded to the nearest 100 by our modified averaging algorithm.

		time to convergence ($noff|_{n=2400:10:2500}$)						
			noise variance					
I.C.	training seed†	BEP	0.000625	0.00125	0.0025	0.005	0.0087	0.015
I	a	720	700	700	800	900	(0.5)	(1.3)
	b		700	700	800	1100	2200	(1.7)
	c		700	700	800	1100	2100	(1.1)
II		890	900	900	900	1100	2300	(1.4)
	b		900	900	1000	1200	(3.4)	(4.5)
	c		900	900	1000	1100	(0.5)	(1.3)
III		960	1000	1000	1000	1300	2100	(1.3)
	b		1000	1000	1100	1300	(0.5)	(4.2)
	c		1000	1100	1100	1800	2200	(1.3)
IV		920	900	1000	1100	1200	(0.8)	(1.9)
	b		1000	1000	1000	1200	2200	(2.0)
	c		1000	1000	900	1300	2200	(1.1)

†trseed values for a, b, and c are 492012, 398092, and 832919, respectively. Due to the increased spike size, the averaging method is based around 0.3 rather than 0.2.



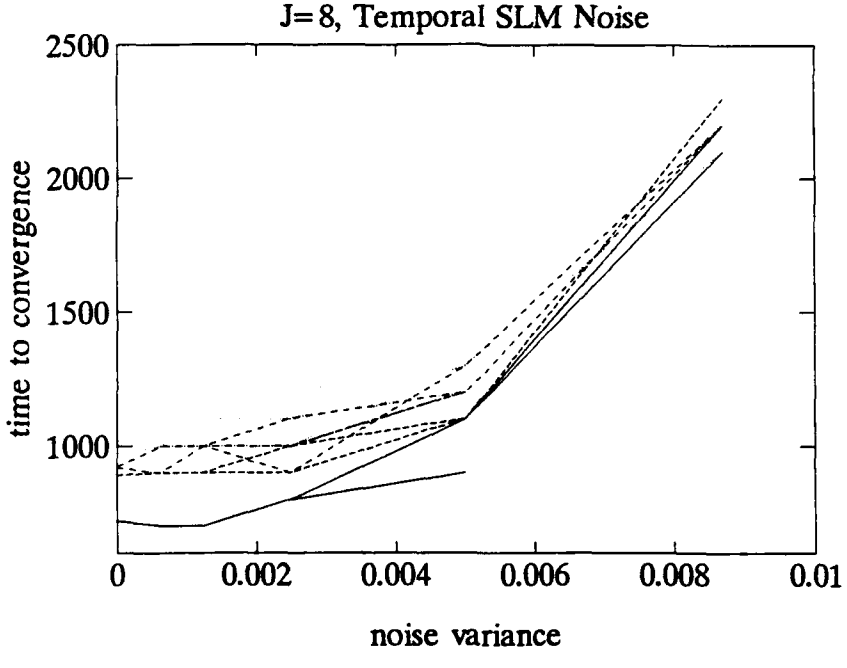


Figure 23. Convergence time vs. SLM temporal noise variance for two networks with different hidden layer sizes. Three traces are shown per initial condition set—for the three training seeds.

c. Space-variant Gain

The results suggest that the larger network is more immune to space-variant gain than is the smaller. See Table XVII. Gain variances up to 0.05 did not prevent convergence, whereas in the $J = 4$ case (Table III), even variances of 0.025 occasionally did. Figure 24 illustrates the significant increase in allowable gain variance that occurs with the increased hidden layer size.

TABLE XVII

Simulation results with nonuniformity in the SLM electro-optic response, for $J = 8$.

		time to convergence $(\text{noff})_{n=2400:10:2500}$						
			variance of element response multipliers					
I.C.	seed†	BEP	0.003125	0.00625	0.0125	0.025	0.05	0.087
I	a	720	720	720	700	710	800	(3)
	b		720	730	730	770	880	(3)
	c		710	690	640	600	550	730
II	a	890	870	860	910	1220	1880	(2)
	b		880	900	1160	1350	1390	(3)
	c		890	880	1390	1920	1560	(0↔4)
III	a	960	960	840	740	700	680	(0)
	b		970	920	830	770	840	2080
	c		1020	1050	950	880	990	810
IV	a	920	930	1010	1030	940	1160	(3)
	b		1020	1010	890	770	670	1000
	c		950	940	920	880	1100	1170

†svseed values for a, b, and c are 573487, 998853, and 893289, respectively.

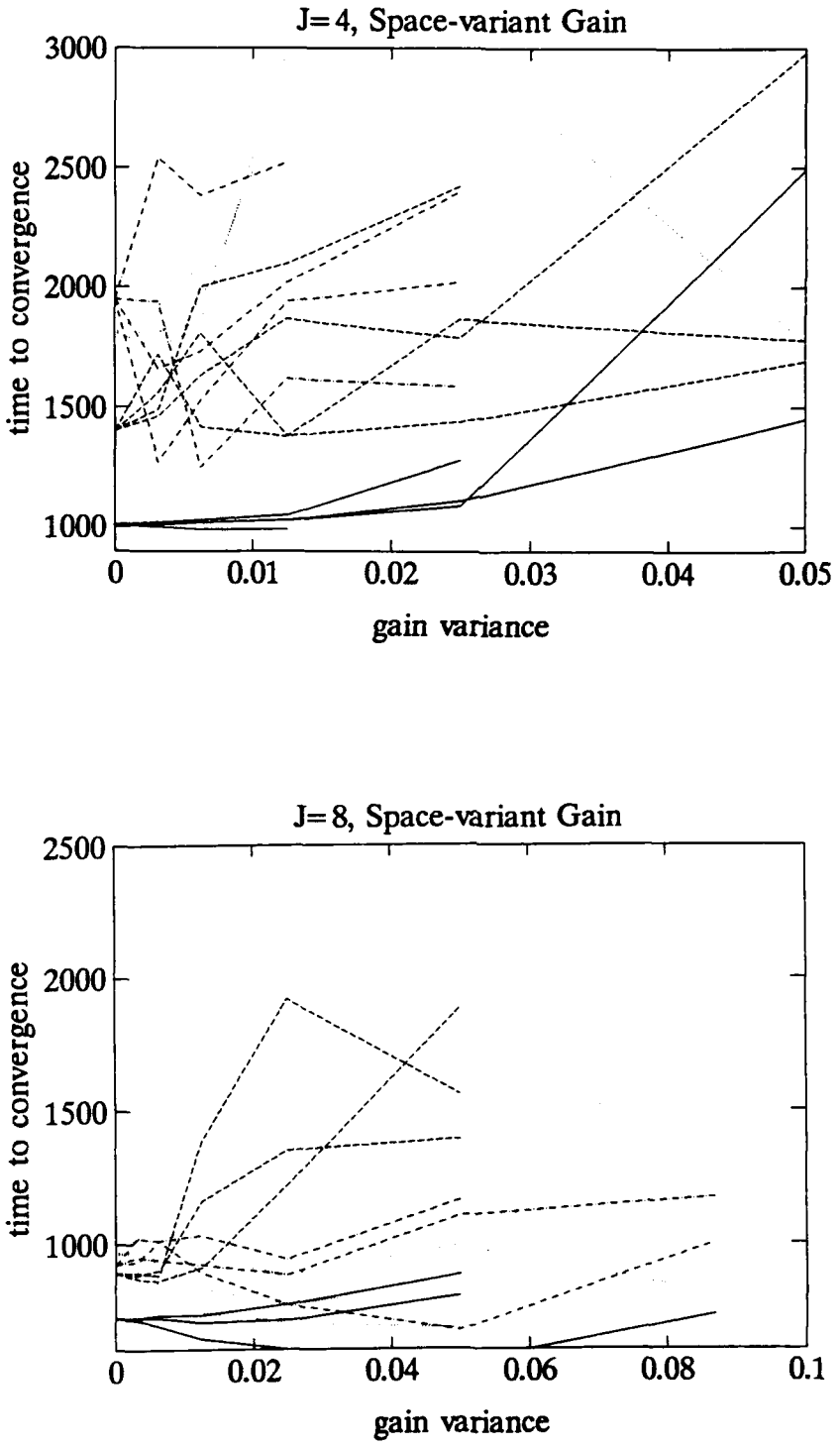


Figure 24. Convergence time vs. space-variant gain for two networks with different hidden layer sizes. In the $J = 4$ case are shown four traces per initial condition set—for the four sets of multipliers s.

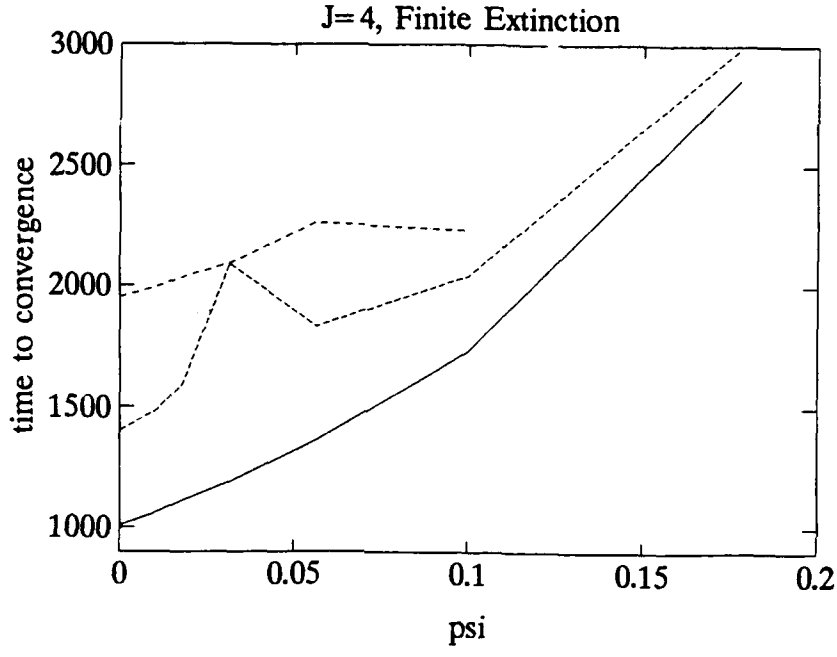
d. Finite Extinction Ratio

As our results show, the size of the hidden layer seems to have little effect on the change in time to convergence resulting from decreased extinction ratio in the polarizers. Table V and Table XVIII show the results for $J = 4$ and $J = 8$, respectively. Figure 25 shows that increasing J does slightly decrease the rate of change in convergence time with ψ—as well as increase the probability of convergence at higher ψ values.

TABLE XVIII
Simulation results with finite extinction ratio in the SLMs, for $J = 8$.

	time to convergence								
	extinction ratio								
I.C.	∞	100	56.23	31.62	17.18	10	5.623	3.162	1.778
I	720	750	770	830	920	1130	1680	>2500	[stuck]
II	890	930	960	1010	1130	1340	1940	>2500	[stuck]
III	960	1000	1030	1090	1200	1460	2170	>2500	[stuck]
IV	920	960	1000	1060	1180	1450	2090	>2500†	[stuck]

†It is uncertain whether this trial was on its way to convergence.



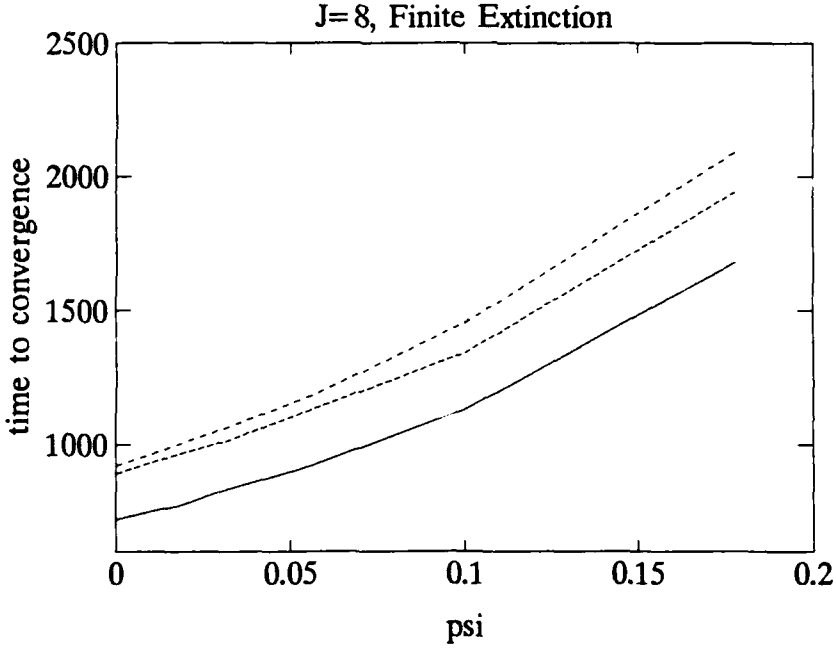


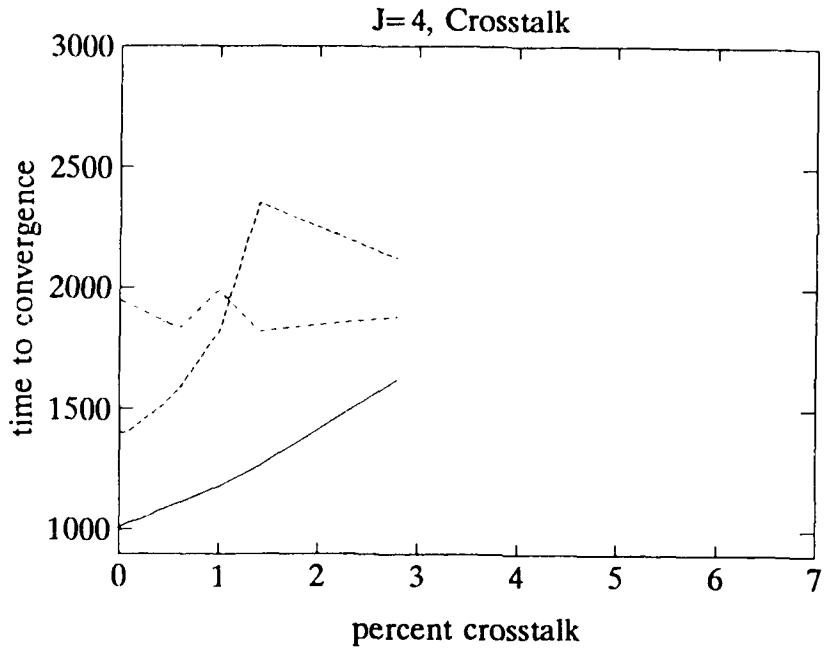
Figure 25. Convergence time vs. ψ (the reciprocal of extinction ratio) for two networks with different hidden layer sizes.

2. The Optical Imaging System

In the $J = 4$ simulations (Table VI), we found that large enough values of crosstalk had unpredictable effects on convergence time. We also found that, above 6% the convergence was effectively rendered improbable. For $J = 8$, the effect is a more predictable increase in convergence time with added crosstalk, as can be seen in Table XIX. Larger values, above 6%, were tolerable. Figure 26 shows graphically the results for both J values.

TABLE XIX
The effects of crosstalk upon convergence, for $J = 8$.

	time to convergence									
	crosstalk									
I.C.	0%	0.05%	0.1%	0.2%	0.4%	0.8%	1.6%	3.2%	6.4%	12.8%
I	720	720	720	730	740	770	830	970	1240	[stuck]
II	890	890	900	910	910	940	1000	1290	1240	[stuck]
III	960	960	980	980	990	1000	1030	1130	1410	[stuck]
IV	920	910	910	910	910	950	980	1130	1330	[stuck]



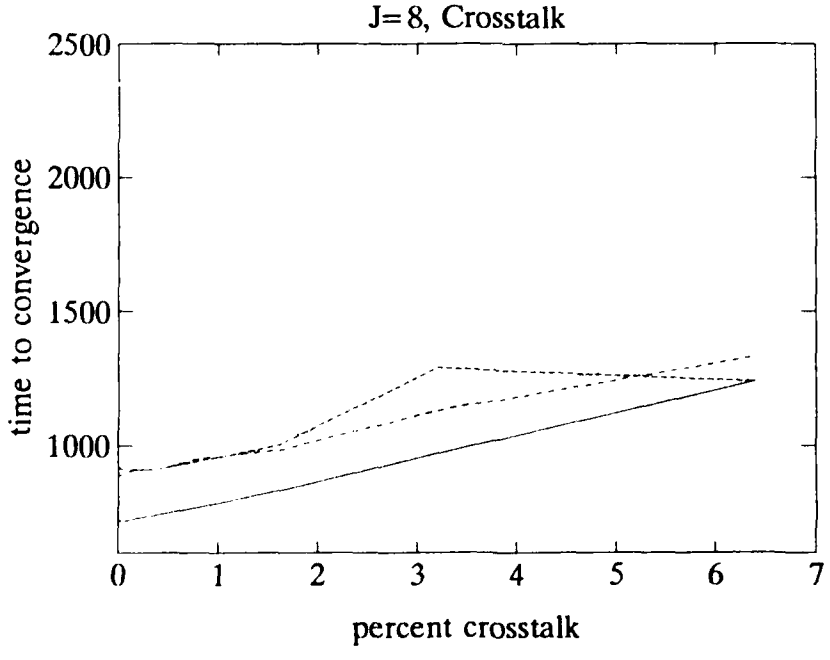


Figure 26. Convergence time vs. percent crosstalk for two networks with different hidden layer sizes.

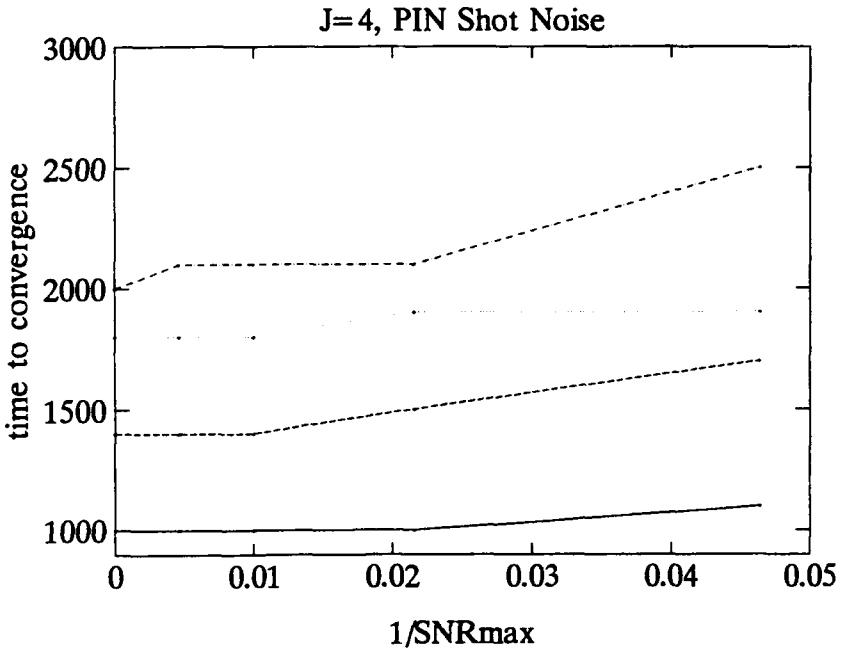
3. The Detectors

a. PIN Shot Noise

Table XX shows the simulation results. As revealed in Figure 27, there is little difference from the $J = 4$ case in the allowable SNR_{max} range. Interestingly, though, unlike in the $J = 4$ case, the learning curves of *noff* are remarkably spike-free. The averaging metric was not needed, therefore, so that the values in the Table are not all divisible by 100.

TABLE XX
Shot noise in the PIN detectors, for $J = 8$.

	time to convergence, $(noff]_{n=2400:10:2500}$						
	SNR_{max}						
I.C.	∞	215.4	100	46.42	21.54	10	4.462
I	720	710	720	710	810	(1.1)	(6.9)
II	890	900	900	900	890	(1.3)	(6.1)
III	960	960	960	960	1010	(1.3)	(7.1)
IV	920	920	920	920	990	(1.0)	(5.8)



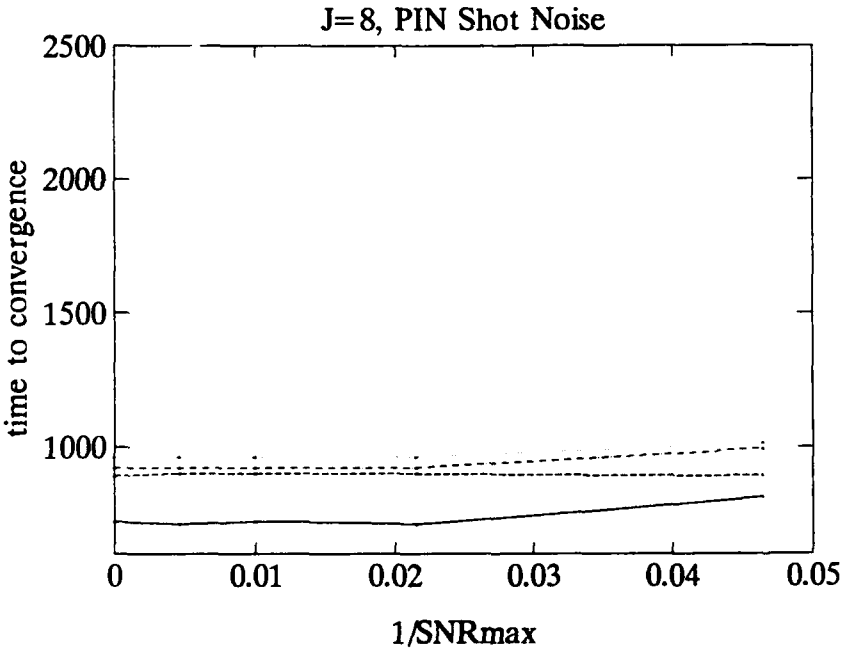


Figure 27. Convergence time vs. PIN shot noise for two networks with different hidden layer sizes.

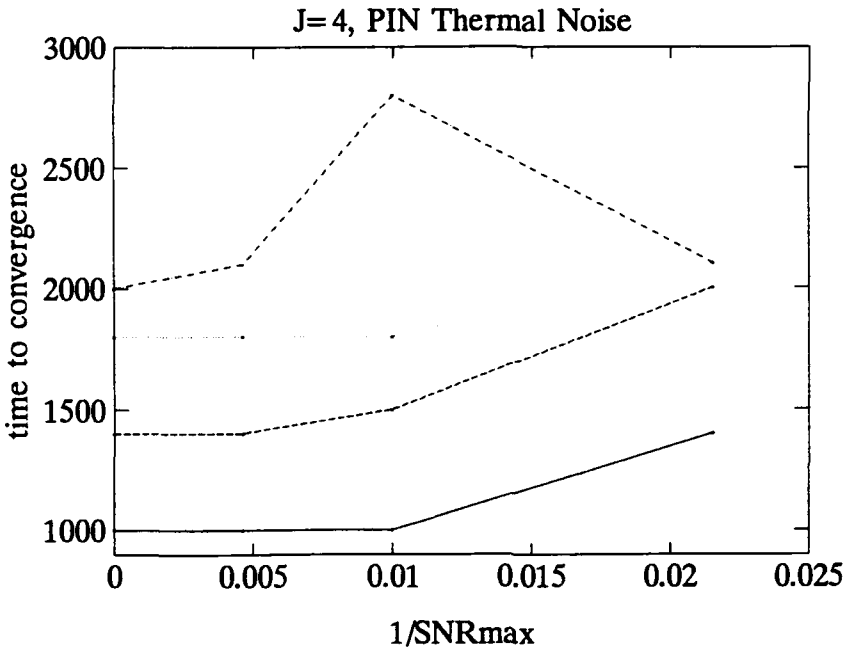
b. PIN Thermal Noise

Table XXI shows the simulation results. Here again, little difference in the allowable SNR_{max} range is created by having four additional hidden units. This is confirmed in Figure 28.

TABLE XXI

Thermal noise in the PIN detectors, for $J = 8$. Note that, except for the case of ∞, values are rounded to the nearest 100 by our de-spiking algorithm.

	time to convergence, $(noff]_{n=2400:10:2500}$						
	SNR_{max}						
I.C.	∞	215.4	100	46.42	21.54	10	4.462
I	720	700	700	1300	(2.2)	(9.1)	(11.0)
II	890	900	900	1300	(2.8)	(9.3)	(11.5)
III	960	1000	1000	1300	(1.5)	(9.1)	(11.6)
IV	920	900	1000	1300	(1.6)	(9.2)	(11.1)



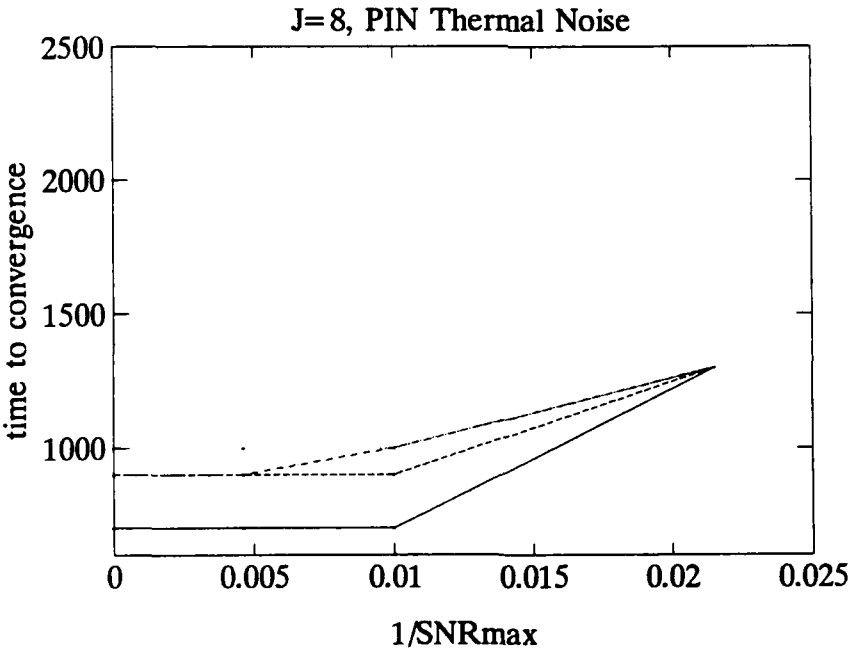


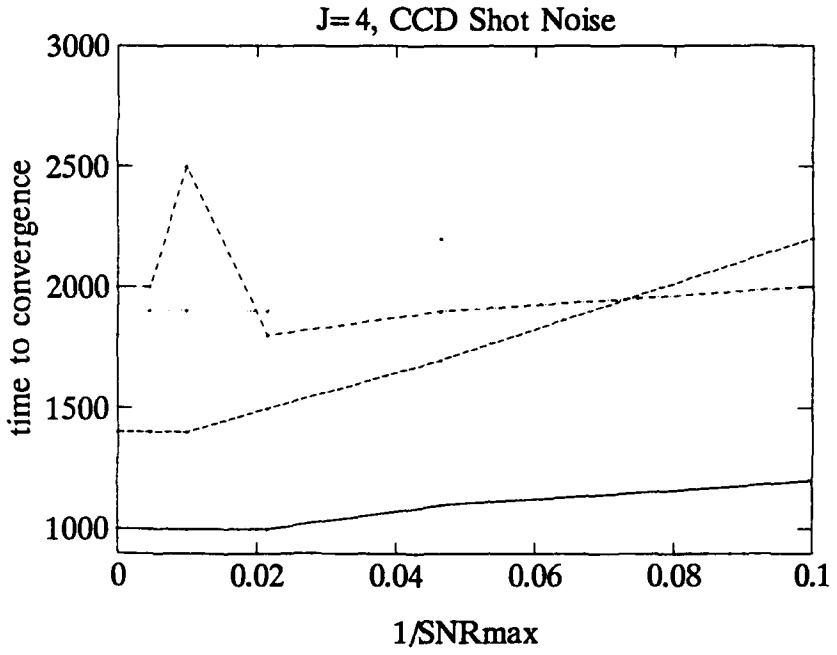
Figure 28. Convergence time vs. PIN thermal noise for two networks with different hidden layer sizes.

c. CCD Shot Noise

Table XXII shows the simulation results. Although this is confirmed by only one trial, convergence is possible for an SNR_{max} as low as 4.462 for $J = 8$, unlike for $J = 4$. Figure 29 shows the graphical comparison.

TABLE XXII
Shot noise in the CCD detectors, for $J = 8$.

	time to convergence, $(noff]_{n=2400:10:2500}$						
	SNR_{max}						
I.C.	∞	215.4	100	46.42	21.54	10	4.462
I	720	720	720	720	750	700	1690
II	890	890	890	890	920	910	[stuck]
III	960	960	970	990	1000	820	(8.2)
IV	920	920	910	890	880	610	(3.5)



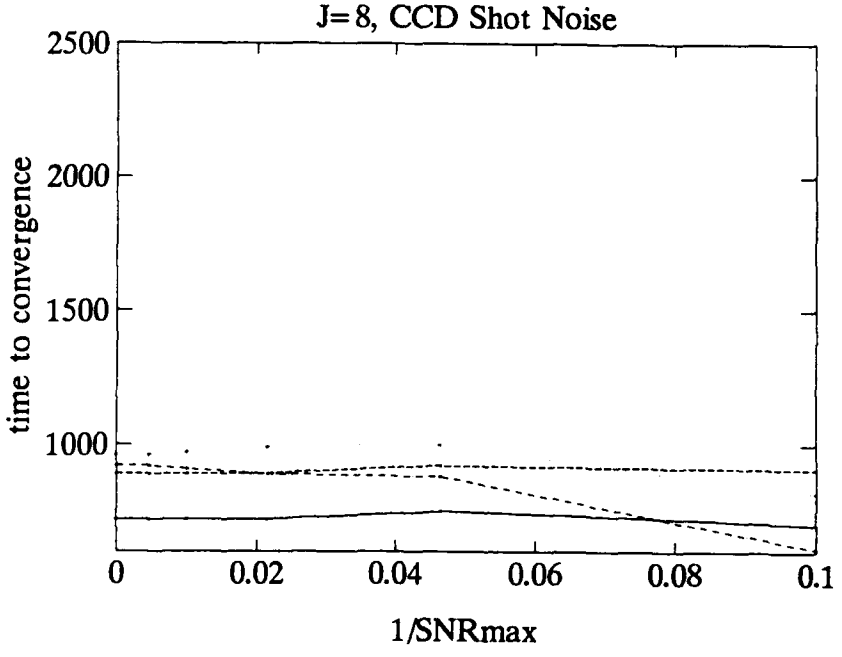


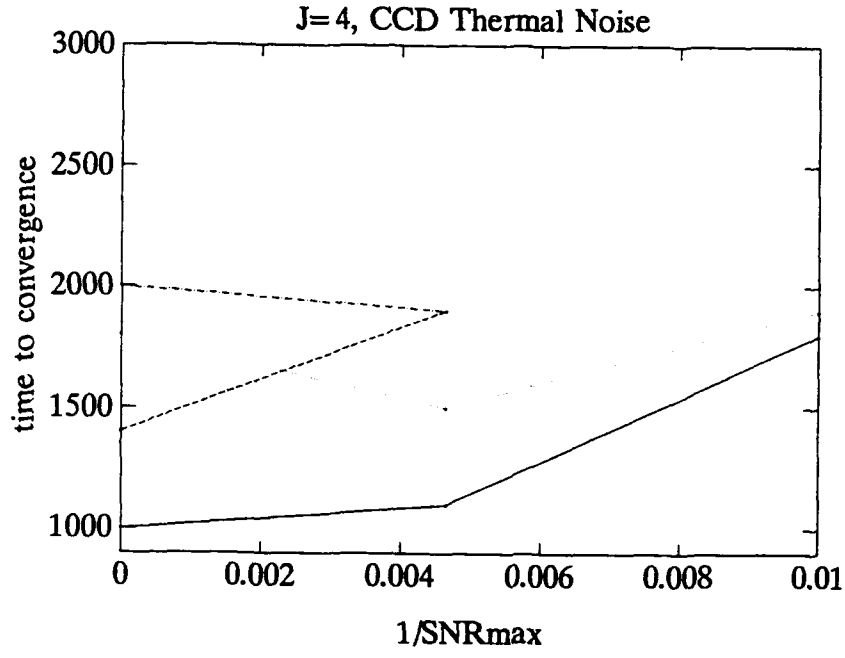
Figure 29. Convergence time vs. CCD shot noise for two networks with different hidden layer sizes.

d. CCD Thermal Noise

Table XXIII shows the simulation results. Having $J = 8$ can allow an SNR_{max} as low as 100. Figure 30 shows the graphical comparison.

TABLE XXIII
Thermal noise in the CCD detectors, for $J = 8$.

	time to convergence, $(noff]_{n=2400:10:2500}$						
				SNR_{max}			
I.C.	∞	215.4	100	46.42	21.54	10	4.462
I	720	710	610	(6.3)	(11.3)	(12.9)	(12.2)
II	890	920	1620	[stuck]	[stuck]	[stuck]	[stuck]
III	960	1100	950	(9.1)	(11.6)	(12.9)	(12.2)
IV	920	800	600	(6.3)	(11.8)	(12.9)	(12.2)



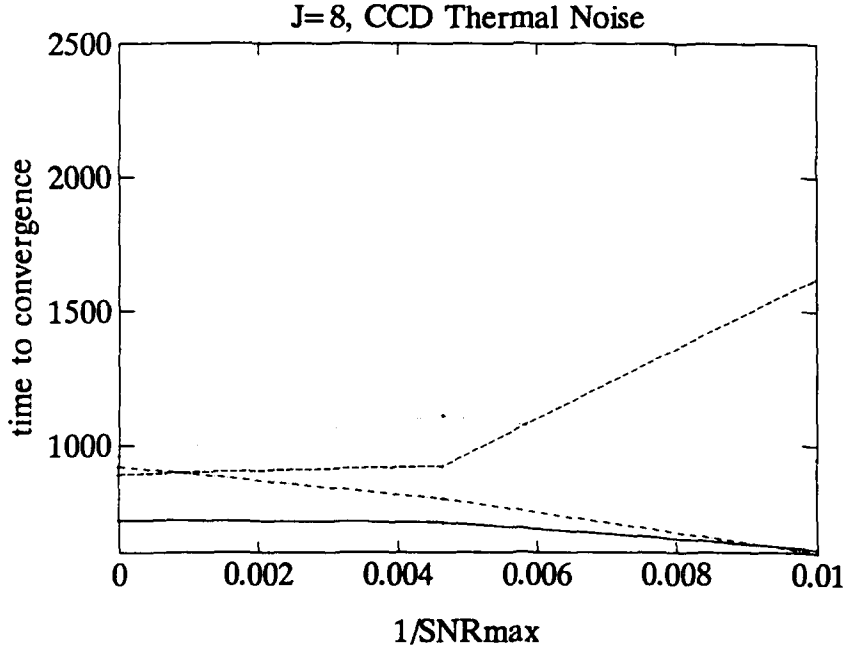


Figure 30. Convergence time vs. CCD thermal noise for two networks with different hidden layer sizes.

E. Combinations of Key Effects (J = 8)

We performed simulations similar to those combinations we had run with $J = 4$.

1. Temporal Noise and Malus's Law in the Weights

As was the case for temporal noise by itself, the learning curve spikes were large enough to require using 0.3 in the averaging metric in place of 0.2. As in the $J = 4$ case, a sweet spot is manifest for each of the traces.. The results are given in Table XXIV. Figure 31 shows the sweet spots graphically.

TABLE XXIV

The combining of the Malus's Law nonlinearity with SLM temporal noise,† for $J = 8$.

(time to convergence, $(noff]_{n=2400:10:2500}$)

	bias				
variance	0	0.1	0.2	0.3	0.4
0.005	2000	1400	2300	2400	2500
0.0025	1700	1000	900	1100	2400
0.00125	1700	1000	900	800	1100

†I.C. 1, trseed=205464.

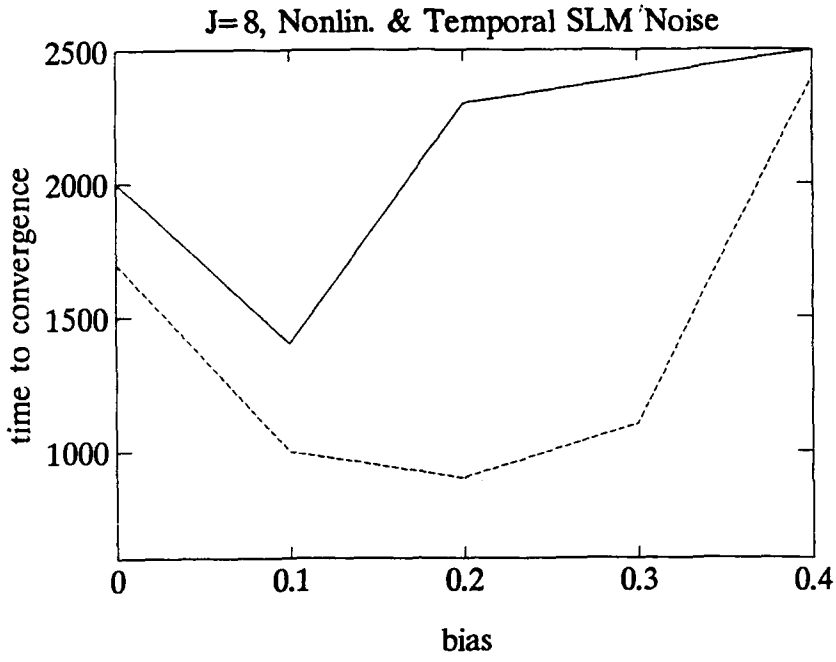


Figure 31. The existence of a "sweet spot" in bias, whereat the bias's benefit and harm are in balance. The solid curve represents a noise variance of 0.005; the dashed curve, 0.0025; the dotted curve, 0.00125.

2. Crosstalk and Shot Noise

As was the case for $J = 4$, (See Table XII.) crosstalk and shot noise effects appear to accumulate. This time, however, spikes do appear, and so the averaging method, with 0.2, was used. The results are shown in Table XXV.

TABLE XXV
The combining of crosstalk and shot noise†, for $J = 8$.

I.C.	time to convergence BEP	minimal‡	maximal††
I	720	800	2500
II	890	900	1300
III	960	1000	1300
IV	920	1000	1000

†trseed=606842
‡0.2% crosstalk, SNR_{max}(PIN shot) = 100, SNR_{max}(CCD shot) = 46.42
††3.2% crosstalk, SNR_{max}(PIN shot) = 21.54, SNR_{max}(CCD shot) = 10.

3. The Complete Detector Noise Models

The results shown in Table XXVI suggest that the degree to which the different types of noise detrimentally interact is not significantly reduced by hidden layer redundancy. The equivalent $J = 4$ results were presented in Table XIII.

TABLE XXVI
The complete detector noise models†, for $J = 8$.

I.C.	time to convergence BEP	minimal‡	maximal††
I	720	760	[stuck]
II	890	960	[stuck]
III	960	960	[stuck]
IV	920	880	[stuck]

†trseed=606842

‡SNR_{max}(PIN shot) = 100, SNR_{max}(CCD shot) = 46.42, SNR_{max}(PIN thermal) = 100, SNR_{max}(CCD thermal) = 464.2.

††SNR_{max}(PIN shot) = 21.54, SNR_{max}(PIN thermal) = 46.42, SNR_{max}(CCD shot) = 10, SNR_{max}(CCD thermal) = 100.

4. The Complete, Compensated SLM Noise Model

The data are shown in Table XXVII. The "minimal" effects do not seem to form a lethal combination. However, the learning curves for the "maximal" conditions took on an extremely spiked appearance. While in most cases, a few spurious spikes after "convergence" are forgiven, in this case we judged the amount to be unacceptable. Figure 32 shows one of the maximal learning curves. Plainly, the excessive spiked appearance makes the use of an averaging metric a mere exercise.

TABLE XXVII
The complete, compensated SLM noise model, for $J = 8$.†

I.C.	time to convergence BEP	minimal‡	maximal††
I	720	700	[spiked]
II	890	900	[spiked]
III	960	940	[spiked]
IV	920	1000	[spiked]

†trseed=205464; svseed=573487.
‡$\sigma^2 = 0.00125$, $\sigma^2(s) = 0.004$, ext. ratio = 215.4.
††$\sigma^2 = 0.005$, $\sigma^2(s) = 0.01$, ext. ratio = 17.78.

This problem did not occur for the $J = 4$ case. It seems that one price paid for hidden layer redundancy is in the sensitivity of the system after a convergence of sorts has already occured.

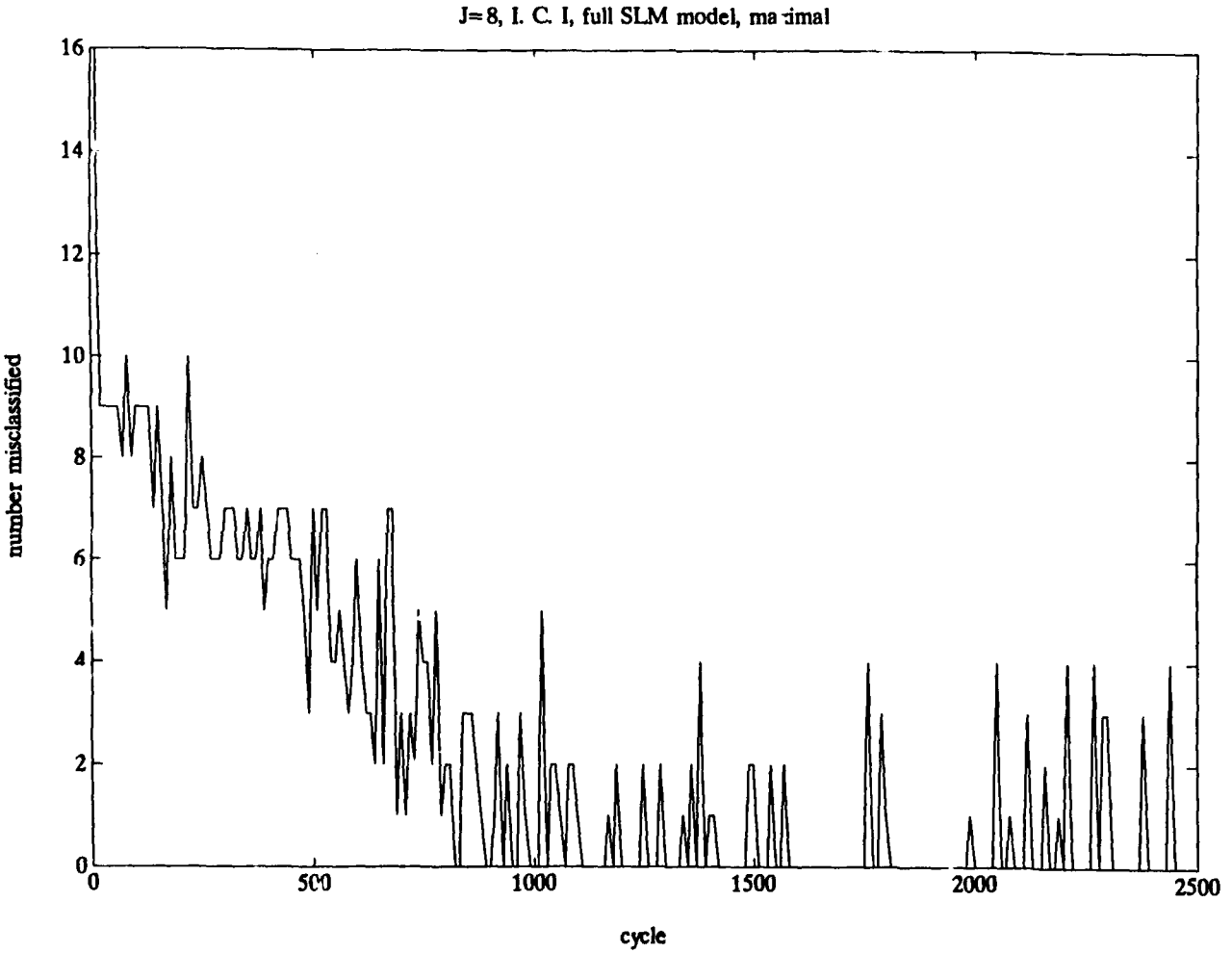


Figure 32. Learning curve for simulation with the complete SLM noise model, with relatively large parameters.

F. The Entire Architecture

We performed several sets of simulations in which all imperfections were assumed to exist. To allow for the likely possibility that predistortion compensation for the SLM nonlinearity would be used, we simulated the network with predistortion enabled and disabled, using a bias of $t = 0.15$ in the latter case. Naturally, the values of all the other parameters must be chosen with care.

Let us begin with the SLMs. With the current technology in Pockel's-cell devices, keeping temporal noise in the driving electronics below 0.2% is achievable. Space-variant gain is a function of fabrication uniformity, primarily in thickness. This can be controlled typically to within a percent. Extinction ratios on dichroic sheet polarizers are typically 10^4; based on our results concerning extinction ratio, and given imperfect alignment, we allowed extinction ratios to be only 250.

Crosstalk is an easily-prevented occurence; its prevention simply requires increased inter-element spacing. We assumed that 0.2% would always be present.

Our PIN detector noise values were obtained using an analysis like that at the end of Section IV. B. 3. For these simulations, we used SNR_{max}(PIN shot) = 1500, and SNR_{max}(PIN thermal) = 250.

For the CCDs, based on currently available technology, we set SNR_{max}(CCD shot) at 500, and SNR_{max}(CCD thermal) at 1000.

For these trials we allowed the simulations to run longer than we had for the previous ones. That is, for $J = 4$, we set N = 4000; and for $J = 8$, we set $N = 3500$. Tables XXVIII and XXIX show the results for four trial sets. Tables XXX and XXI repeat the same data, except that each trial's simulation time is shown divided by the corresponding "no effects" time. This makes interpreting the data easier.

The "trial 1" data for both J values show, for the most part, an increase in convergence time. In trial 2, we kept all parameters the same except for temporal noise, whose variance we increased to 0.005. We felt that this increased convergence time by what might might be considered excessive; in some cases, the convergence time doubled, for $J = 4$. So we backed it down.

TABLE XXVIII
Simulation of the entire architecture, for $\dot{J} = 4$.†

I.C.	nonlinearity	time to convergence: no effects	trial 1	trial 2	trial 3	trial 4
I	compensated	1010	1300	2100	1400	[stuck]
	biased	1160	1500	3300	1600	[stuck]
II	compensated	1400	2900	2200	2300	2200
	biased	1520	1800	2800	2000	2300
III	compensated	1810	3300	2500	2900	1500
	biased	1720	1900	3600	1900	4000
IV	compensated	1950	2000	2500	1700	2900
	biased	1800	1800	3100	2100	1900
	SLM temp.		0.002	0.005	0.002	0.002
	s.-v. gain		0.01	0.01	0.01	0.01
	ext. ratio		250	250	100	30
	crosstalk		0.002	0.002	0.005	0.01
	PIN shot		1500	1500	1500	1500
	PIN thermal		250	250	250	250
	CCD shot		500	500	500	250
	CCD thermal		1000	1000	1000	500

†trseed=490001; svseed=598392.

TABLE XXIX
Simulation of the entire architecture, for $J = 8$.†

I.C.	nonlinearity	time to convergence: no effects	trial 1	trial 2	trial 3	trial 4
I	compensated	720	800	1000	800	1200
	biased	830	900	1250	1000	1100
II	compensated	890	1000	1200	1000	1100
	biased	1140	1100	1250	1000	1000
III	compensated	960	900	1300	900	1200
	biased	900	800	1250	900	1100
IV	compensated	920	1200	1300	1000	1200
	biased	840	1300	1250	1000	2100
	SLM temp.		0.002	0.005	0.002	0.002
	s.-v. gain		0.01	0.01	0.01	0.01
	ext. ratio		250	250	100	30
	crosstalk		0.002	0.002	0.005	0.01
	PIN shot		1500	1500	1500	1500
	PIN thermal		250	250	250	250
	CCD shot		500	500	500	250
	CCD thermal		1000	1000	1000	500

†trseed=492012; svseed=573487.

TABLE XXX
Simulation of the entire architecture, for $\dot{J} = 4$.†

I.C.	nonlinearity	time to convergence divided by that for no effects: no effects	trial 1	trial 2	trial 3	trial 4
I	compensated	1	1.29	2.08	1.39	[stuck]
	biased	1	1.29	2.84	1.64	[stuck]
II	compensated	1	2.07	1.57	1.64	1.57
	biased	1	1.18	1.84	1.32	1.51
III	compensated	1	1.82	1.38	1.60	0.83
	biased	1	1.10	2.09	1.10	2.33
IV	compensated	1	1.03	1.28	0.87	1.49
	biased	1	1.00	1.72	1.17	1.06
		SLM temp.	0.002	0.005	0.002	0.002
		s.-v. gain	0.01	0.01	0.01	0.01
		ext. ratio	250	250	100	30
		crosstalk	0.002	0.002	0.005	0.01
		PIN shot	1500	1500	1500	1500
		PIN thermal	250	250	250	250
		CCD shot	500	500	500	250
		CCD thermal	1000	1000	1000	500

†trseed=490001; svseed=598392.

TABLE XXXI
Simulation of the entire architecture, for $J = 8$.†

I.C.	nonlinearity	time to convergence divided by that for no effects: no effects	trial 1	trial 2	trial 3	trial 4
I	compensated	1	1.11	1.39	1.11	1.67
	biased	1	1.08	1.51	1.20	1.33
II	compensated	1	1.12	1.35	1.12	1.24
	biased	1	0.96	1.10	0.88	0.88
III	compensated	1	0.94	1.35	0.94	1.25
	biased	1	0.89	1.39	1.00	1.22
IV	compensated	1	1.30	1.41	1.09	1.30
	biased	1	1.55	1.49	1.19	2.50
		SLM temp.	0.002	0.005	0.002	0.002
		s.-v. gain	0.01	0.01	0.01	0.01
		ext. ratio	250	250	100	30
		crosstalk	0.002	0.002	0.005	0.01
		PIN shot	1500	1500	1500	1500
		PIN thermal	250	250	250	250
		CCD shot	500	500	500	250
		CCD thermal	1000	1000	1000	500

†trseed=492012; svseed=573487.

In trial 3, we increased crosstalk and worsened the extinction ratio. For the most part, the results are quite similar to those from trial 1. Evidently, these changes, particularly the one in crosstalk, play a less significant part than a similar change in SLM temporal noise.

In trial 4, we further worsened extinction ratio and crosstalk, and significantly reduced the CCD performance. Again, these conditions seemed to have a similar effect to increasing SLM temporal noise, but this time some of the $J = 4$ runs failed to converge.

In conlusion, an optical back propagation network solving a problem of the size of 2-D corners will perform best with extra hidden units, and will require conditions at least as good as those set in trial 2 or 3.

V. Exemplar-based Model

The BEP model for a neural net classifier separates the input patterns by a series of hyperplanes. As we illustrated in Section III. B. 2., each of the hyperplanes is associated with a hidden processing element and the orientation and location of the hyperplane is completely specified by the weight vector associated with that processing element. One or more layers of hidden processing elements are needed to form an arbitrary decision boundary via intersections of half spaces.[19]

Exemplar-based classifiers[20] use their hidden processing elements somewhat differently. The weight vectors associated with the processing elements in the first layer simply represent the input patterns (exemplars). The subsequent layers then group outputs from the first layer processing elements and make a classification decision. The subsequent processing differs between k-nearest neighbor classifiers,[21,22] Restricted Coulomb Energy classifiers[23,24] or Radial Basis Function classifiers.[25,26] The common characteristic of these models is that they have high storage requirements (proportional to the exemplars in the training set) but short training times. The error signals in the training cycle are used mainly to adjust the weights in the subsequent layers or some simple parameters (e.g., the size of the basin of attraction) associated with the processing elements in the first layer. Below are some descriptions of those exemplar-based networks that we have studied and simulated.

A. The Nearest-neighbor Network

The simplest exemplar-based approach is to assign to every training pair an exemplar center. The task of the network would then be to determine which of these "neighbors" is the nearest to the given input. It then outputs the class of the nearest neighbor. There is no training step, of course, other than the initialization.

The distance computation is based around a vector-matrix multiplication, not unlike a hyperplane-based network. Let an I-element input vector o_i be presented, and the position vector of the jth hidden unit be C_{ji}. Then the distance is

$$|o_i - C_{ji}|^2 = |o_i|^2 + |C_{ji}|^2 - 2\, o_i \cdot C_{ji}$$

and J such distances exist. However, since $|o_i|^2$ will be the same for all hidden units, all that is needed for comparison is $|C_{ji}|^2 - 2\ o_i \cdot C_{ji}$. After computing these J comparative distance terms, the network passes only the output value associated with that hidden unit whose distance equals the minimum.

Figure 33 shows how a nearest-neighbor network allocates the decision space for the 2-D skewed corners problem. Also shown is the network's solution when nine of the boundary data points are missing. It is interesting to note that back propagation (with $J = 4$) was unable to solve this sparse version of the corners problem. While the network's generalization is good, the efficiency of hidden unit allocation is obviously low. One solution might be to use only selected inputs as exemplars.

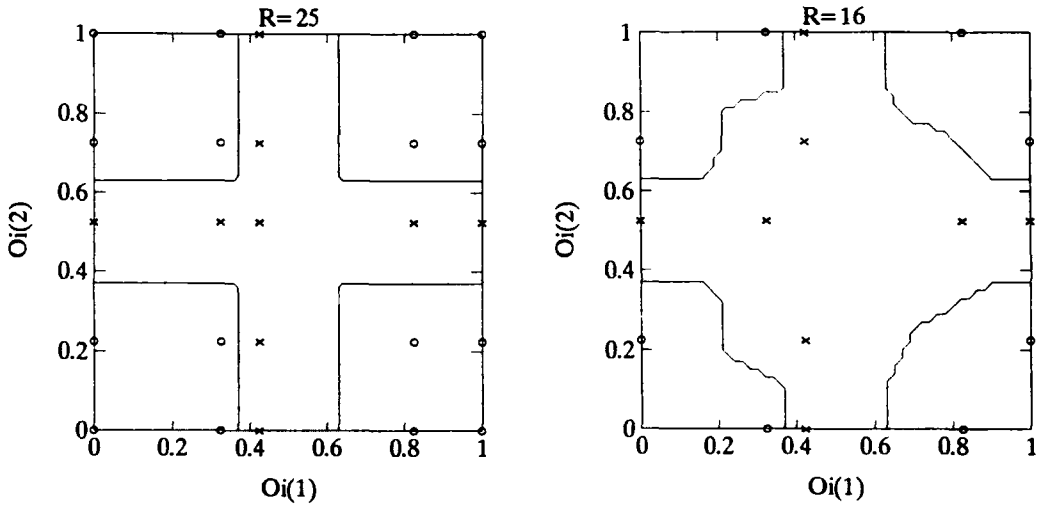


Figure 33. The decision regions produced by a nearest-neighbor network for the 2-D skewed corners problem.

B. The Nestor Learning System

The Nestor Learning System, based on Restricted Coulomb Energy methods[23,24], is one system that we have studied. This method dynamically commits new hidden units (called prototype units) as training progresses, and modifies existing ones, until all training pairs are correctly classified.

The neighborhoods in this system are hypersherical in shape; their centers are themselves co-located with selected training inputs. Each neighborhood is tagged with

one of the possible output classes of the network. As each training pair is presented, the algorithm

- reduces the size of those neighborhoods which overlap and conflict with that pair, and

- commits a new prototype unit, centered at that pair, if the pair is unclassified (that is, if it is located inside *no* neighborhoods).

Naturally, the order of presentation of the training data strongly determines where the neighborhoods will end up; this can give rise to inefficient configurations, where a large portion of the input space is unidentified or confused. One advantage of this method is that very few cycles of training data presentation are needed before the network is solved. It is important to note that what we have here described, somewhat tersely, is what Nestor Corporation calls a single-layer system (despite the multilayer nature)—the version that is marketed by Nestor in fact uses a sophisticated multilayer configuration.

We wrote a Matlab simulation program for this system. The program is given in the Appendix, Section C. Figure 34 shows the decision space allocation of a solved network for the 2-D skewed corners problem with $J = 15$. In this case, the presentation order was left-to-right, beginning with the bottom row on the graph. While only three epochs were needed to solve the problem, the poor generalization is self-evident.

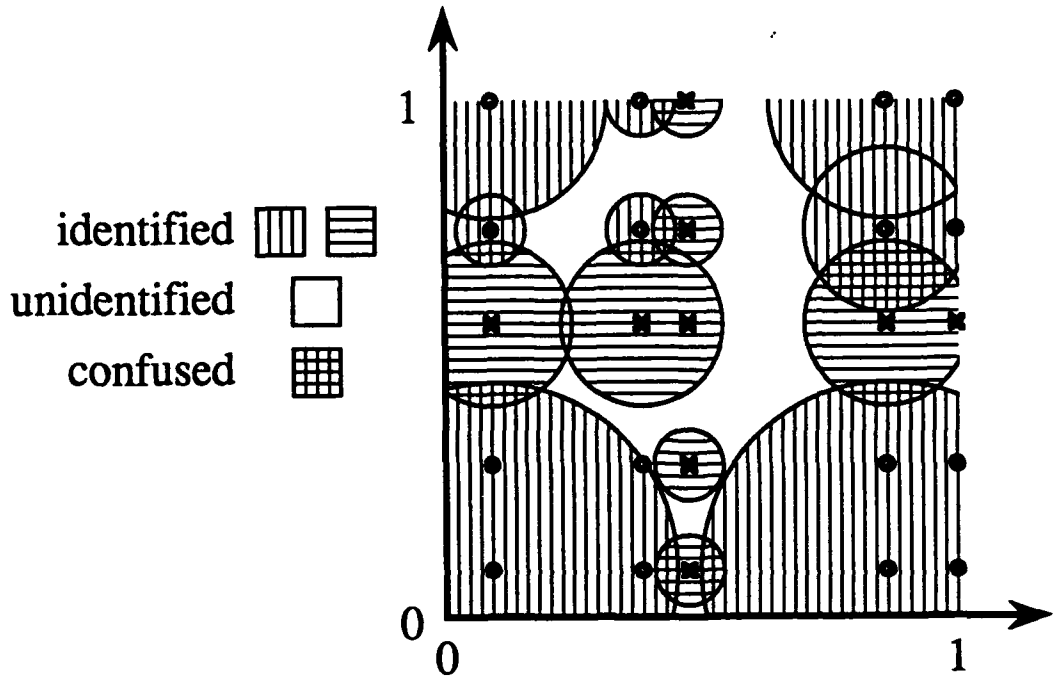


Figure 34. The decision regions produced by a Nestor Learning System single-layer network. The network produces identified, unidentified, and confused reponses to inputs in the shown regions.

C. Modified BEP

As mentioned in Figure 1, we at BDM have developed an algorithm which uses a gradient descent to effect learning in a radial basis function network. Recall Section III. B. 1., in which the forward pass equations and the delta rule were used to derive equations for the weight and bias updates. The same type of derivation can be used for a network with different forward pass equations. In a radial basis function network, the hidden units produce outputs which exponentially fall off with increasing input distance from a centroid. The rate of falloff varies with direction; therefore, surfaces of constant output are ellipsoids rather than simply hypersheres. The output units form hyperplanes, just as in traditional BEP. Thus, the forward pass equations are:

Forward Pass

1. $Net_j = \sum_i (o_i - C_{ji})^2/\sigma_{ji}^2$ $\qquad o_j = \exp(-Net_j)$

2. $Net_k = \sum_j o_i W_{kj}$ $\qquad o_k = 1/(1 + \exp(-Net_k))$

where C_{ji} represents the locations of the centers, W_{kj} the weights in the output layer, and σ_{ji} is a constant used to shape the ellipsoids.

Recall the delta rule:

$$\Delta W \propto -\partial E/\partial W_{kj} \text{ where } E = \sum_{r} \sum_{k} (t_{rk}-o_{rk})^2$$

Since the equations for Net_k and o_k are identical to those in conventional BEP, so will be the equations for δ_k and ΔW_{kj}. For the hidden layer the delta rule calls for $\partial E/\partial C_{ji}$, which can be computed using the chain rule:

$$\partial E/\partial C_{ji} = (\partial E/\partial Net_j) \times (\partial Net_j/\partial C_{ji}).$$

We can show that

$$\partial Net_j/\partial C_{ji} = -2\ (o_i-C_{ji})\ /\ \sigma_{ji}^2.$$

This one term is the only term in the sum over all i that is nonconstant with respect to C_{ji}. The other link we will continue to call $-\delta_j$:

$$\partial E/\partial Net_j = -\delta_j = (\partial E/\partial o_j) \times (\partial o_j/\partial Net_j)$$

of which $\partial o_j/\partial Net_j$ is just $-o_j$. The first link is

$$\partial E/\partial o_j = \partial/\partial o_j \{ \sum_{k} (\partial E/\partial Net_k)(\partial Net_k/\partial o_j) \}$$

$$= -\sum_{k} \delta_k\ \partial/\partial o_j(\sum_{k} W_{kj}o_j) = -\sum_{k} \delta_k\ W_{kj}$$

Therefore,

$$\delta_j = -\sum_{k} \delta_k\ W_{kj}\ o_j$$

and

$$\Delta C_{ji} = -2 \lambda \delta_j (o_i - C_{ji}) / \sigma_{ji}^2.$$

where λ is a learning rate constant, similar to η.

At this point the hidden ellipsoid sizes σ_{ji} remain to be computed. Beginning with the chain rule,

$$\partial E/\partial \sigma_{ji} \qquad j \qquad j \quad ji \qquad \partial E/\partial \sigma = (\partial E/\partial Net) \times (\partial$$

We can show that

$$\partial Net_j/\partial \sigma_{ji} = -2 (o_i - C_{ji})^2 / \sigma_{ji}^3.$$

The other link appeared in the hidden center discussion and is the same $-\delta_j$. The delta rule then rewritten:

$$\Delta \sigma_{ji} \propto -\partial E/\partial \sigma_{ji} = -2 \xi \delta_j (o_i - C_{ji})^2 / \sigma_{ji}^3$$

where ξ is yet a third learning rate variable. In summary, the delta rule yields up the following equations for updating the output weights, hidden centers, and hidden sizes:

Gradient Descent Equations

3. $\delta_k = o_k (1 - o_k) (t_k - o_k)$
4. $\delta_j = - \sum_k \delta_k W_{kj} o_j$
5. $\Delta W_{kj} = \eta o_j \delta_k$
6. $\Delta C_{ji} = -2 \lambda \delta_j (o_i - C_{ji}) / \sigma_{ji}^2$
7. $\Delta \sigma_{ji} = -2 \xi \delta_j (o_i - C_{ji})^2 / \sigma_{ji}^3$

In the above equations, the learning rates are given as three different variables—η, λ, and ξ, because they appear in differing types of equations. However, there is no reason why hyperplane-based back propagation could not also possess different learning rates for the different layers.

We coded modified back propagation in the Matlab modules given in the Section D of Appendix. Figure 35 shows the learning curve of one trial of the system, where 8 hidden units are used; Figure 36 shows the corresponding solution space. Notice that only 180 iterations were required.

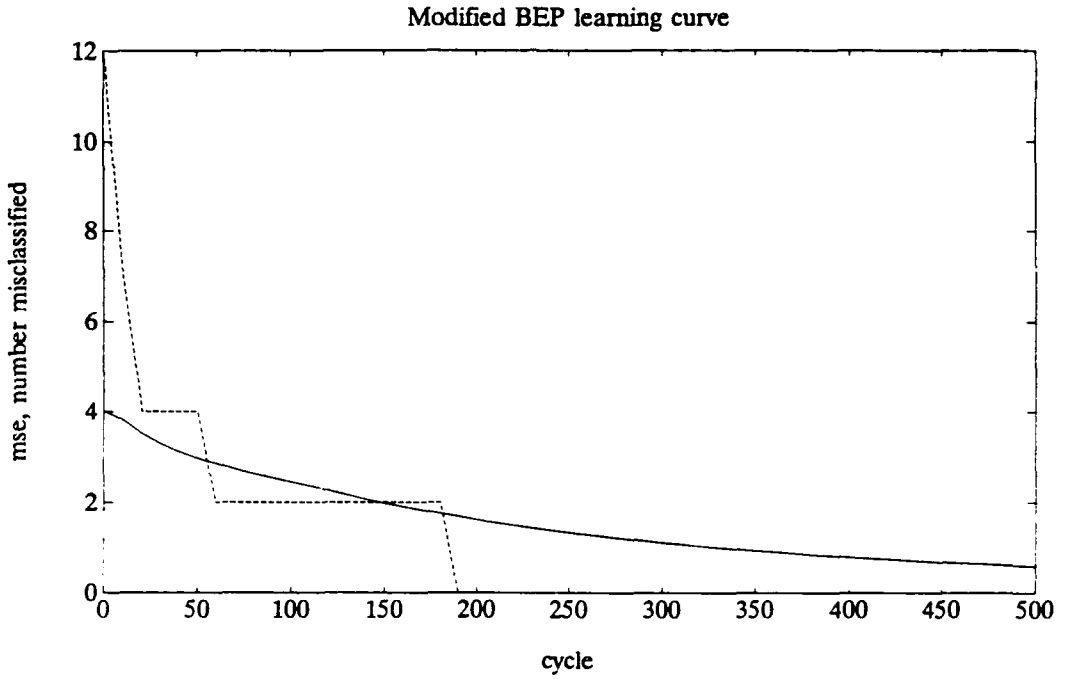


Figure 35. A learning curve produced by modified BEP on the 2-D skewed corners problem, with $J = 8$.

The solution initially surprised us; rather than drawing nearly circular ellipses about the corner regions, the network discovered the horizontal and vertical rows of the other class. Also, the network gives no regard to the exact locations of the corner training data—outside the narrow ellipses. (By comparison, note Figure 8; the 0.5 output contour curve generated by back propagation is located halfway between the locations of the two classes, at all corners.) That the generated solution depends on only some of the data suggests that the generalization of this network is poor.

Note that the ellipsoidal neighborhoods are restrained to having major axes along one of the input space axes. However, one could begin with a more elaborate set of forward pass equations, based on ellipsoidal regions which can be tilted. It is interesting to speculate what solutions such a network would generate.

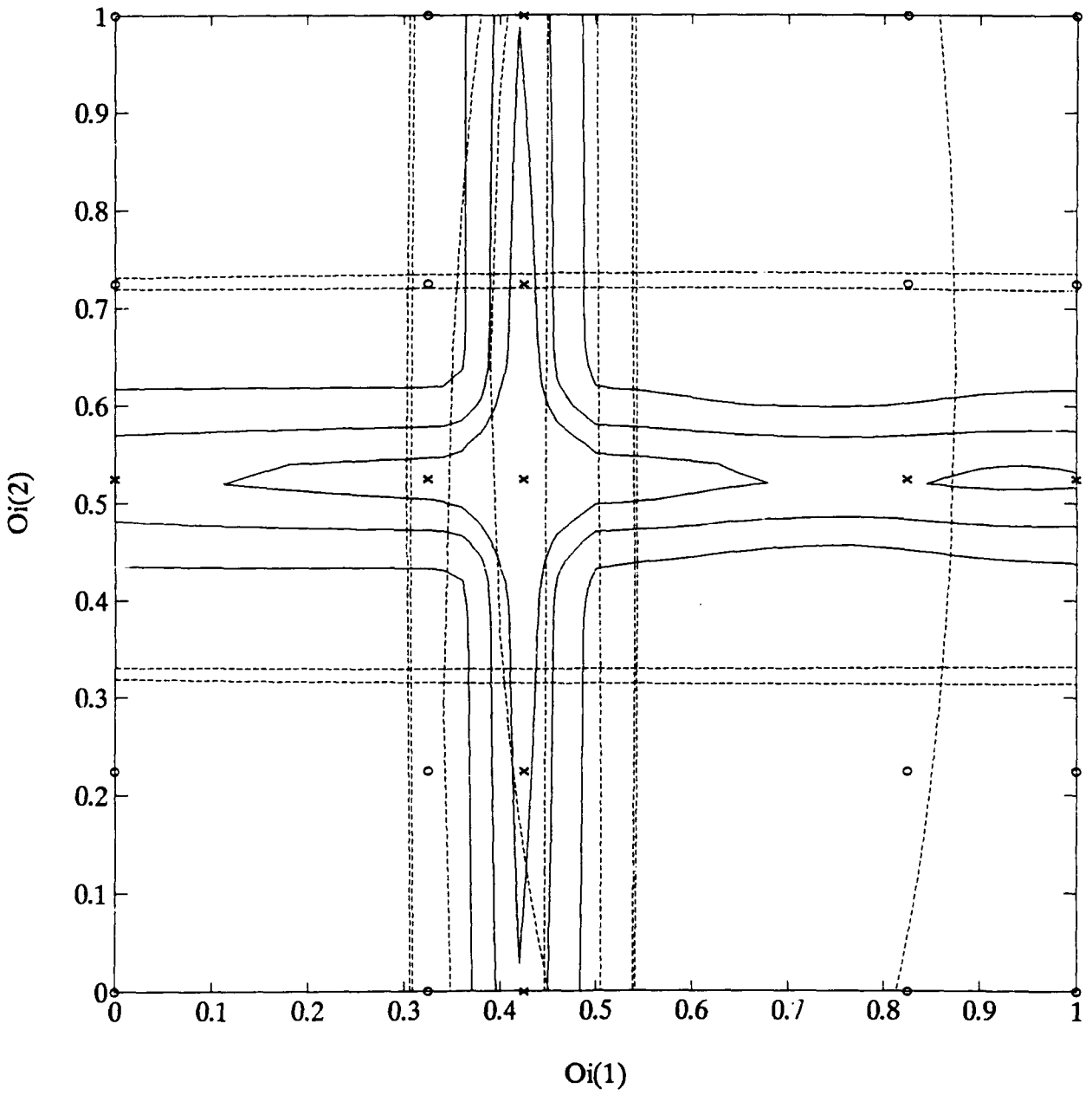


Figure 36. The decision regions produced by modified BEP. The hidden decision bounds are horizontally and vertically oriented ellipses. Solid lines show a contour plot of the output, similar to that in Figure 7.

VI. DISCUSSION AND CONCLUSIONS

Our analysis of an optical back propagation architecture was preceded by a study of the algorithm itself and a technology-independent analysis. In the process, we discovered an important and usually-overlooked initialization scheme developed by Sheldon Gilbert. This technique, as well as other enhancements to the original back propagation, shows promise in significantly reducing the number of cycles to convergence. We have also studied neighborhood-based multilayer feedforward nets. In particular, we proposed a novel approach to learning in one such network using radial basis functions. This, too, shows room for enhancement, though as it is convergence time is already reduced compared to back propagation.

The technology-dependent analysis of optical neural networks has yielded a wealth of useful information. Our requirement of full utilization of the 3-D nature of optical communications for all operations that warrant it has resulted in the design of a promising arhcitecture. We have analyzed this architecture in a part-by-part fashion, developing a rigorous model for the SLMs, imaging system, and detection systems.

While excessive noise in a given element may prevent convergence our simulations have revealed relatively little "noise accumulation." That a learning curve can exhibit both a noisy appearance and convergence is evidence of the robustness of the back propagation algorithm. However, our simulations also show us that the presence of an uncorrected nonlinearity due to Malus's law will significantly degrade performance. Introducing a bias is a simple remedy to this problem; nevertheless, it will also be necessary to insure that noise levels are kept low for this remedy to be useful. This may not be achievable with present SLM analog driving technology; predistortion compensation would then be required.

We have also observed that performance and noise immunity are both greatly increased by the use of extra hidden units. Further, extra hidden units are not only easy to add in an optical architecture; it is also likely they will already exist. That is, the number of hidden units for a given complex problem is usually not known *a priori*, so that extra hidden units will have to be made available.

In general, we have shown that an optical back propagation architecture is capable of full operation with realistic hardware. This is especially true for the case of

hidden layer redundancy, and where the SLM nonlinearity has been predistortion-compensated.

VII. REFERENCES

1. T. Kohonen, *Self-Organization and Associative Memory*, New York: Springer-Verlag, 1984.

2. S. Grossberg, *Studies of Mind and Brain*, Boston, MA: Reidel, 1982.

3. M. Caudill and C. Butler, Eds., *Proceedings of the IEEE First International Conference on Neural Networks*, vol. I-IV, San Diego, June 21-24, 1987.

4. M. Takeda and J. W. Goodman, "Neural Networks for Computation: Representations and Programming Complexity," Appl. Opt. **25**, 3033-3046 (1986).

5. A. D. Fisher, W. L. Lippincott, and J. N. Lee, "Optical Implementations of Associative Networks with Versatile Adaptive Learning Capabilities," Appl. Opt. **23**, 5039-5054 (1987).

6. D. Brady, X. G. Gu, and D. Psaltis, "Photorefractive Crystals in Optical Neural Computers," Proc. SPIE **882**, 20 (1988).

7. J. J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," Proc. Natl. Acad. Sci **79**, 2554-2558 (1982).

8. D. Psaltis and N. Farhat, "Optical Information Processing Based on an Associative Memory Model of Neural Nets with Thresholding and Feedback," Opt. Lett. **10**, 98 (1985).

9. Special Issue on Neural Networks, Appl. Opt. **26**, Dec. 1988.

10. M. Minsky and S. Papert, *Perceptrons*, Cambridge, MA: MIT Press, 1969.

11. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA: MIT Press, 1986, pp. 318-362.

12. B. Widrow and M. E. Hoff, "Adaptive Switching Circuits," *1960 IRE WESCON Convention Record*, New York: IRE, pp. 96-104.

13. R. P. Gorman and T. Sejnowski, "Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets," Neural Networks, **1**, 75-89, (1988). See also IEEE Trans. ASSP **36**, 1135-1140 (1988).

14. G. Bradshaw, R. Fozzard, and L. Ceci, "A Connectionist Expert System that Actually Works," Adv. Neural Inf. Proc. Syst. **1**, 248-251 (1988).

15. K. Wagner and D. Psaltis, "Multilayer Optical Learning Networks," Appl. Opt. **26**, 5061-5076 (1987).

16. M. Kranzdorf, B. J. Bigner, L. Zhang, and K. M. Johnson, "Optical Connectionist Machine with Polarization-based Bipolar Weight Values," Opt. Engg. **28**, 844-848 (1989).

17. A. Von Lehmen, E. G. Paek, P. F. Liao, A. Marrakchi, and J. S. Patel, "Influence of Interconnection Weight Discretization and Noise in an Opto-electronic Neural Network," Opt. Lett. **14**, 928-930 (1989).

18. S. L. Gilbert, "Implementing Artificial Neural Networks in Integrated Circuitry: A Design Proposal for Back-Propagation," Lincoln Laboratory, MIT, Lexington, MA, Technical Report 810, Nov. 1988.

19. R. P. Lippmann, "An Introduction to Computing with Neural Nets," IEEE ASSP Magazine, pp. 4-22, April 1987.

20. R. P. Lippmann, "Pattern Classification Using Neural Networks," IEEE Communications Magazine, pp. 47-64, November 1989.

21. S. M. Omohundro, "Efficient Algorithms with Neural Network Behavior," Complex Systems 1, 273-347 (1987).

22. C. Stanfill and D. Waltz, "Toward Memory-based Reasoning," Commun. of the ACM **29**, 1213-1228 (1986).

23. D. L. Reilly, L. N. Cooper, and C. Elbaum, "A Neural Model for Category Learning," Bio. Cybernetics **45**, 35-41 (1982).

24. G. V. Puskorius and L. A. Feldkamp, "A Characterization and Emulation of the Nestor Learning System," Ford Motor Company, Dearborn, MI, Technical Report SR-88-126, December 1988.

25. D. S. Broomhead and D. Lowe, "Radial Basis Functions, Multi-Variable Functional Interpolation and Adaptive Networks," Royal Signals and Radar Establishment, Malvern, Worcester, Great Britain, Memorandum 4148, March 1988.

26. S. Renals and R. Rohwer, "Phoneme Classification Experiments using Radial Basis Functions," *Proc. Int'l Joint Conf. on Neural Networks*, Washington, DC: IEEE, pp. I.461-I.467, June 1989.

VIII. KEY PERSONNEL

A. Dr. Michael W. Haney

Dr. Haney is a principal staff member and the director for optical computing technology at BDM International, Inc. and the program manager on this contract. He received his Ph.D. in electrical engineering in 1986 from the California Institute of Technology. His dissertation was: "Acousto-optical Time-and-Space Integrating Processors for Real-Time Synthetic Aperture Radar Imaging."

B. Dr. Ravindra A. Athale

Dr. Athale, former manager for optical computing technology at BDM, is currently an associate professor of electrical engineering at George Mason University and continues his involvement in this program as a regular consultant. He received his Ph.D. in electrical engineering in 1980 from the University of California at San Diego. The title of his dissertation was: "Studies in Digital Optical Computing."

C. Mr. James J. Levy

Mr. Levy is an associate staff member for optical computing technology at BDM International, Inc. He received his M.S. in electrical engineering in 1987 from the University of Delaware. His thesis title was: "Grating Couplers for Optical Waveguides."

IX. CONFERENCE PRESENTATIONS

James J. Levy, Ravindra A. Athale, and Michael W. Haney presented a paper at the Annual Meeting of the Optical Society of America, held November 4-9, 1990 in Boston. Below are given the abstract and summary, which appeared in the advance program and technical digest, respectively.

Abstract

Computer simulations reveal the effects of noise in optical weight matrix elements, updated by back propagation, upon the learning curve characteristics.

Summary

The back propagation algorithm[1] has become increasingly popular in the neural net research community. Various optical implementations have been proposed, with the hope of increased performance via the parallelism of optics. Realistic models of opto-electronic implementations must include the effects of noise. While noise often "anneals," increasing the convergence rate, excessive noise can effectively transform any updating algorithm into a random search among weight configurations. For the purpose of evaluating the effects of component noise on the performance of the back propagation algorithm, we have developed a simulation program which allows insertion of noise terms wherever appropriate. The program introduces noise processes into the weight updating process during the learning phase. The learning curve is defined as the mean-square error vs. iteration number; our analysis emphasizes examination of learning curves rather than simply probability of convergence. In this paper we describe the behavior and noise tolerance of back propagation as related to hidden layer size, learning rate, and initial conditions.

1. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in Parallel Distributed Processing, vol. 1, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA: M.I.T. Press, 1986, pp. 318-362.

APPENDIX: PROGRAM LISTINGS

Below are given the input and output .mat files representing the 2-D skewed corners problem.

bpin.mat		bpout.mat
0	0	0.1
0.325	0	0.1
0.425	0	0.9
0.825	0	0.1
1	0	0.1
0	0.225	0.1
0.325	0.225	0.1
0.425	0.225	0.9
0.825	0.225	0.1
1	0.225	0.1
0	0.525	0.9
0.325	0.525	0.9
0.425	0.525	0.9
0.825	0.525	0.9
1	0.525	0.9
0	0.725	0.1
0.325	0.725	0.1
0.425	0.725	0.9
0.825	0.725	0.1
1	0.725	0.1
0	1	0.1
0.325	1	0.1
0.425	1	0.9
0.825	1	0.1
1	1	0.1

A. Straight Back Propagation

In choosing a problem for simulation, we look for one which is not prone to local minima. Having chosen the problem, we also fine tune the learning rate and the parameters to be used by the Gilbert initialization process. We are not unaware that in the real world, this luxury is usually not available; nevertheless, we are evaluating not algorithms, but architectures. Therefore we are justified in fine tuning the solution process and thereby making the simulations run more easily.

This procedure requires the performance of many runs quickly. Even with all the imperfections disabled, the full-blown simulation uses sign encoding, thereby more

than doubling the computation. We therefore used a straight back propagation series, the b series, for quick evaluations as these. The b series consists of "bframe.m," which prompts for the various parameters; "binit.m," which loads the data and sets up the initial state of the network; and "bctrain.m," which effects the training procedure.

Any function calls which are not part of PC-Matlab are listed in this Appendix.

bframe.m

```
% bframe script
disp('This lets you run binit and btrain by having')
disp('you put in all the necessary variables beforehand.')
J=input('Enter J:  ');
icseed=input('Enter icseed:  ');
smartinit=input('Enter 1 to initialize by method of Gilbert:  ');
if smartinit==1
   steepness=input('Enter the initial hidden unit hill steepness:  ');
   spread=1;
else
   spread=input('Enter the initial condition spread:  ');
end
eta=input('Enter eta:  ');
N=input('Enter N:  ');
DN=input('Enter DN:  ');
msetol=input('Enter the mse below which training stops:  ');
a=input('Enter the upper (and -lower) bound:  ');
```

binit.m

```
% binit script
load bpin
load bpout
I=ncols(bpin)
K=ncols(bpout)
R=nrows(bpout)
if exist('keepWji')
   Wji=keepWji; THETAj=keepTHETAj;
   Wkj=keepWkj; THETAk=keepTHETAk;
else
   % Initialize the weights and biases
   % random numbers about zero center
   rand('uniform')
   rand('seed',icseed)
   rand(1,3); % to exercise generator
   Wji=(rand(I,J)-1 ./2)*spread;
   THETAj=(rand(1,J)-1 ./2)*spread;
   Wkj=(rand(J,K)-1 ./2)*spread;
   THETAk=(rand(1,K)-1 ./2)*spread;
   if smartinit==1
      % the Wji
```

```
      % numerator is predet. magnitude of weights vector
      % which's normal to dividing hyperplane (line for I=2)
      x=steepness./sqrt(sum(Wji.^2));
      Wji=(ones(I,1)*x).*Wji;
      % the THETAj
      THETAj=-0.5*sum(Wji);
      % the Wkj
      Wkj=(1/J)*ones(J,K);
   end
   keepWji=Wji, keepTHETAj=THETAj
   keepWkj=Wkj, keepTHETAk=THETAk
end
n=0;
% MSE CALCULATION
% Forward Pass
NETj=[bpin,ones(R,1)]*[Wji;THETAj];
Oj=1 ./(1+exp(-NETj));
NETk=[Oj,ones(R,1)]*[Wkj;THETAk];
Ok=1 ./(1+exp(-NETk));
ldetmse(1)=sum(meanc((Ok-bpout).^2));
% literally, less detailed mse
% this first one's not an average
noff(1)=sum(sum((abs(Ok-bpout)>0.4)));
noff2(1)=sum(sum((abs(Ok-bpout)>0.2)));
clear ans x % temporary variables
```

bctrain.m

```
% bctrain script
% Accumulates weight change over epoch, THEN updates...
stop=n+N;
while n<stop
   n=n+1
   % Error back propagation
   Dk=(bpout-Ok).*Ok.*(1-Ok);
   DWkj=eta*Oj'*Dk;
   DTHETAk=eta*ones(1,R)*Dk;
   Dj=Oj.*(1-Oj).*(Dk*Wkj');
   DWji=eta*bpin'*Dj;
   DTHETAj=eta*ones(1,R)*Dj;
   % Addition of the deltas to electronic register
   Wkj=Wkj+DWkj; THETAk=THETAk+DTHETAk;
   Wji=Wji+DWji; THETAj=THETAj+DTHETAj;
   % Only clipping remains...
   Wkj=-(Wkj<-a).*a+(abs(Wkj)<=a).*Wkj+(Wkj>a).*a;
   THETAk=-(THETAk<-a).*a+(abs(THETAk)<=a).*THETAk+(THETAk>a).*a;
   Wji=-(Wji<-a).*a+(abs(Wji)<=a).*Wji+(Wji>a).*a;
   THETAj=-(THETAj<-a).*a+(abs(THETAj)<=a).*THETAj+(THETAj>a).*a;
   % Forward Pass
   NETj=[bpin,ones(R,1)]*[Wji;THETAj];
   Oj=1 ./(1+exp(-NETj));
   NETk=[Oj,ones(R,1)]*[Wkj;THETAk];
   Ok=1 ./(1+exp(-NETk));
   detmse=[detmse,sum(meanc((Ok-bpout).^2))];
```

```
   if rem(n,DN)==0
      ldetmse(n/DN+1)=mean(detmse);
      noff(n/DN+1)=sum(sum((abs(Ok-bpout)>0.4)));
      noff2(n/DN+1)=sum(sum((abs(Ok-bpout)>0.2)));
      clear detmse
      disp('convergence check')
      % based on mse over r, not individuals
      if ldetmse(n/DN+1)<=msetol
         disp('less detailed mse within tolerance')
         break
      end
   end
end
clear stop Dk Dj % temporary variables
```

meanc.m

```
function s=meanc(in)
% out=meanc(in)
%
% This function returns a column vector with the mean of
% each row of in.  This includes the case where in has but
% one column (i.e., it won't then return the scalar).
% So it's not strictly the transpose of mean.

if ncols(in)==1
   s=in;
else
   s=(mean(in'))';
end
```

ncols.m

```
function nc=ncols(in)
% out=ncols(in)
%
% This function returns the number of columns
% in the matrix passed to it.

[nr,nc]=size(in);
```

nrows.m

```
function nr=nrows(in)
% out=nrows(in)
%
% This function returns the number of rows
% in the matrix passed to it.

[nr,nc]=size(in);
```

We have also developed a diagnostic program, "cpcontor.m", that prints out a contour map of the solved (or unsolved) network, like that shown in Figure 7. This program is valid only where $I = 2$, and generates K plots (one for each output node k).

The program graphs J decision lines, delimiting where o_j crosses 0.5. Given more hidden units than are needed for a problem, back propagation usually deems some as unnecessary and decreases their connections (W_{kj}) to output nodes. On screen, "cpcontor.m" color-codes the decision lines accordingly: those associated with the strongest connections are drawn in white; medium connections yield green decision lines, and weak connections yield blue ones.

The output o_k is represented as a contour map. Red, green, and blue curves correspond to ouputs at 0.3, 0.5, and 0.7, respectively.

Incidentally, running "cpcontor.m" on an untrained net initialized with and without Gilbert's method dramatically illustrates what this method does.

cpcontor.m

```
% cpcontor script
% This program, valid only for two-input nets, generates the
% contour plot of each output unit, hopefully superimposed
% with both the hidden unit decision lines and the input.
load bpin
load bpout
axis([0 1 0 1]) % a more general statement would be
% axis([min(bpin(:,1)) max(bpin(:,1)) min(bpin(:,2)) max(bpin(:,2))])
axis('square')
K=ncols(bpout); R=nrows(bpin);
for k=1:K
   if K>1
      act=['subplot(22'num2str(k) ')']
      eval(act)
   end
   % Inputs
   for r=1:R
      if bpout(r,k)>0.5
         onplot=[onplot;bpin(r,:)];
      else
         offplot=[offplot;bpin(r,:)];
      end
   end
   plot(onplot(:,1),onplot(:,2),'xg'), hold on
   plot(offplot(:,1),offplot(:,2),'og')
   xlabel('Oi(1)'), ylabel('Oi(2)')
   % Dividing lines
```

```
   bgst=max(abs(Wkj(:,k)'));
   for j=J:-1:1
      x=0:0.1:1;
      y=(-x*Wji(1,j)-ones(1,11)*THETAj(j))./Wji(2,j);
      if abs(Wkj(j,k))<(1/3)*bgst
         plot(x,y,'b')
      elseif ((abs(Wkj(j,k))>=(1/3)*bgst)&(abs(Wkj(j,k))<(2/3)*bgst))
         plot(x,y,'g')
      elseif abs(Wkj(j,k))>=(2/3)*bgst
         plot(x,y,'w')
      end
   end
   % Contour
   [xme,yme]=meshdom(0:0.02:1,0:0.02:1);
   for colco=1:51 % because of the 0:0.02:1s above
      NJ=[xme(:,colco),yme(:,colco),ones(51,1)]*[Wji;THETAj];
      OJ=1 ./(1+exp(-NJ));
      NK=[OJ,ones(51,1)]*[Wkj;THETAk];
      OK(:,colco)=1 ./(1+exp(-NK));
   end
   contour(OK,[0.3,0.5,0.7],0:0.02:1,0:0.02:1)
   clear onplot offplot x y X Y NJ NK OJ OK colco j k r xme yme
   hold off
end
axis('normal')
```

B. Optical Back Propagation

The simulation program has been continually evolving since the start of the contract. In the technology-independent phase, additive noise was applied to the weight after each change. Technology-dependent modeling requires making a commitment to a particular architecture, in this case the one in Figure 5. Nevertheless, the basic three-module series approach has been retained. At the end the weights are coded back to bipolar form for use by "cpcontor.m."

Naturally, "uframe.m" must ask the user for many more parameters than "bframe.m" did, most relating to the architecture. Many of the variables that get defined here act as logical variables that will later determine whether imperfection equations are performed or skipped. (Only crosstalk is always computed, even if set to 0.) This skipping over of equations not called for significantly shortens simulation time.

uframe.m

```
% uframe script
% Unlike bframe.m, this is specific to a dedicated subpixel
% architecture, with or without correction for Malus' law.
disp('This lets you run uinit and uctrain by having')
disp('you put in all the necessary variables beforehand.')
J=input('Enter J:  ');
icseed=input('Enter icseed:  ');
smartinit=input('Enter 1 to initialize by method of Gilbert:  ');
if smartinit==1
   steepness=input('   Enter the initial hidden unit hill steepness:  ');
   spread=1;
else
   spread=input('Enter the initial condition spread:  ');
end
eta=input('Enter eta:  ');
N=input('Enter N:  ');
DN=input('Enter DN:  ');
msetol=input('Enter the mse below which training stops:  ');
inject=input('Enter 0 for no noise; 2 for Gaussian noise:  ');
if inject~=0
   trseed=input('   Enter trseed:  ');
end
if inject==2
   var=input('   Enter the variance:  ');
end
disp('You can introduce space-variant gain into the')
disp('voltage-birefringence transfer function.')
svgain=input('Enter 0 not to, 1 to:  ');
if svgain==1
```

```
   svseed=input('   Enter the seed:  ');
   svvar=input('   Enter the variance of this gain:  ');
end
disp('You can then introduce a finite extinction ratio')
disp('to the 1-D SLMs.')
poorext=input('Enter 0 not to, 1 to:  ');
if poorext==1
   extratio=input('   Enter the extinction ratio:  ');
end
disp('You can incorporate sine-related nonlinearities that')
disp('can occur in architectures without electronic')
disp('correction for the law of Malus.')
nonlin=input('Enter 0 not to, 3 for spatial encoding w/ Vbias:  ');
if nonlin==3
   tp2=input('   Enter the tp2 of Vbias:  ');
end
disp('You can introduce PIN detector noise.')
PINnoise=input('Enter 1 to do this:  ');
if PINnoise==1
   if inject==0
      trseed=input('   Enter trseed:  ');
   else
      disp('   The simulation will share the random number')
      disp('   generator with that of weight noise.')
      trseed
   end
   SNR1smax=input('   Enter the maximum SNR permitted by shot noise:  ');
   SNR1t=input('   Enter the SNR permitted by thermal noise:  ');
end
disp('You can introduce CCD detector noise.')
CCDnoise=input('Enter 1 to do this:  ');
if CCDnoise==1
   if ((PINnoise==0)&(inject==0))
      trseed=input('   Enter trseed:  ');
   else
      disp('   The simulation will share the random number')
      disp('   generator with that of weight noise and/or')
      disp('   PIN detector noise.')
      trseed
   end
   SNR2smax=input('   Enter the maximum SNR permitted by shot noise:  ');
   SNR2t=input('   Enter the SNR permitted by thermal noise:  ');
end
hr=input('Enter the upper (and -lower) bound:  ');
if nonlin==3
   disp('   The bias-correcting renormalizing constant')
   disp('   does not affect this bound.')
end
b=input('Enter the Dk normalizing constant (1 or greater):  ');
eps=input('Enter the crosstalk fraction:  ');
```

uinit.m

```
% uinit script
% dedicated subpixel architecture
load bpin
load bpout
I=ncols(bpin)
K=ncols(bpout)
R=nrows(bpout)
if exist('keepWji')
   Wji=keepWji; THETAj=keepTHETAj;
   Wkj=keepWkj; THETAk=keepTHETAk;
else
   % Initialize the weights and biases
   % random numbers about zero center
   rand('uniform')
   rand('seed',icseed)
   rand(1,3); % to exercise generator
   Wji=(rand(I,J)-1 ./2)*spread;
   THETAj=(rand(1,J)-1 ./2)*spread;
   Wkj=(rand(J,K)-1 ./2)*spread;
   THETAk=(rand(1,K)-1 ./2)*spread;
   if smartinit==1
      % the Wji
      % numerator is predet. magnitude of weights vector
      % which's normal to dividing hyperplane (line for I=2)
      x=steepness./sqrt(sum(Wji.^2));
      Wji=(ones(I,1)*x).*Wji;
      % the THETAj
      THETAj=-0.5*sum(Wji);
      % the Wkj
      Wkj=(1/J)*ones(J,K);
   end
   keepWji=Wji, keepTHETAj=THETAj
   keepWkj=Wkj, keepTHETAk=THETAk
end
if svgain==1
   rand('normal')
   rand('seed',svseed)
   % Creation of the svgain matrices
   svWkj=addgauss(ones(J,2*K),svvar)
   svTHETAk=addgauss(ones(1,2*K),svvar)
   svWji=addgauss(ones(I,2*J),svvar)
   svTHETAj=addgauss(ones(1,2*J),svvar)
   svOi=ones(R,1)*addgauss(ones(1,I),svvar) % unipolar
   svOj=ones(R,1)*addgauss(ones(1,J),svvar) %  "   "
   svOk=ones(R,1)*addgauss(ones(1,2*K),svvar)
   svOj=ones(R,1)*addgauss(ones(1,2*J),svvar)
end
if poorext==1, extbias=1/extratio; end
a=hr;
if nonlin==3, a=hr/(mal2a(1-tp2)-mal2a(tp2)); end
% normalizing and encoding (splitting and clipping)
esWkj=encode(Wkj/hr); esTHETAk=encode(THETAk/hr);
```

```
esWji=encode(Wji/hr); esTHETAj=encode(THETAj/hr);
% No bias yet exists to remove
if nonlin==0
   % Inverse Malus's law
   eWkj=imal2a(esWkj); eTHETAk=imal2a(esTHETAk);
   eWji=imal2a(esWji); eTHETAj=imal2a(esTHETAj);
elseif nonlin==3
   eWkj=nencode(decode(esWkj),tp2);
   eTHETAk=nencode(decode(esTHETAk),tp2);
   eWji=nencode(decode(esWji),tp2);
   eTHETAj=nencode(decode(esTHETAj),tp2);
end
% Add just a pinch of noise
if inject==2
   eWkj=addgauss(eWkj,var);
   eTHETAk=addgauss(eTHETAk,var);
   eWji=addgauss(eWji,var);
   eTHETAj=addgauss(eTHETAj,var);
end
% Multiply by space-variant gain
if svgain==1
   eWkj=eWkj.*svWkj; eTHETAk=eTHETAk.*svTHETAk;
   eWji=eWji.*svWji; eTHETAj=eTHETAj.*svTHETAj;
end
% Disallow voltages<0
eWkj=(eWkj<0).*0 + (eWkj>=0).*eWkj;
eTHETAk=(eTHETAk<0).*0 + (eTHETAk>=0).*eTHETAk;
eWji=(eWji<0).*0 + (eWji>=0).*eWji;
eTHETAj=(eTHETAj<0).*0 + (eTHETAj>=0).*eTHETAj;
% Finally, the Malus's law does this:  (no if statement)
oWkj=mal2a(eWkj); oTHETAk=mal2a(eTHETAk);
oWji=mal2a(eWji); oTHETAj=mal2a(eTHETAj);
% And there may be that poor extinction ratio
if poorext==1
   oWkj=(1-extbias)*oWkj+extbias;
   oTHETAk=(1-extbias)*oTHETAk+extbias;
   oWji=(1-extbias)*oWji+extbias;
   oTHETAj=(1-extbias)*oTHETAj+extbias;
end
% Preparation for mse calculation
% the zeroth-iteration column for mse
n=0;
% MSE CALCULATION
oOi=imal2a(bpin);
if inject==2, oOi=addgauss(oOi,var); end
if svgain==1, oOi=oOi.*svOi; end
oOi=(oOi<0).*0 + (oOi>=0).*oOi;
oOi=mal2a(oOi);
if poorext==1, oOi=(1-extbias)*oOi+extbias; end
% Forward Pass
oNETj=vucross([oOi,ones(R,1)],eps)*[oWji;oTHETAj];
oNETj=vcross(oNETj,eps);
if PINnoise==1
   var1=((I+1)*oNETj/SNR1smax^2)+((I+1)*ones(R,2*J)/SNR1t).^2;
```

```
   oNETj=vaddgaus(oNETj,var1);
end
NETj=decode(oNETj);
Oj=1 ./(1+exp(-a*NETj));
oOj=imal2a(Oj);
if inject==2, oOj=addgauss(oOj,var); end
if svgain==1, oOj=oOj.*svOj; end
oOj=(oOj<0).*0 + (oOj>=0).*oOj;
oOj=mal2a(oOj);
if poorext==1, oOj=(1-extbias)*oOj+extbias; end
oNETk=vucross([oOj,ones(R,1)],eps)*[oWkj;oTHETAk];
oNETk=vcross(oNETk,eps);
if PINnoise==1
   var1=((J+1)*oNETk/SNR1smax^2)+((J+1)*ones(R,2*K)/SNR1t).^2;
   oNETk=vaddgaus(oNETk,var1);
end
NETk=decode(oNETk);
Ok=1 ./(1+exp(-a*NETk))
ldetmse(1)=sum(meanc((Ok-bpout).^2));
% literally, less detailed mse
% this first one's not an average
noff(1)=sum(sum((abs(Ok-bpout)>0.4)));
noff2(1)=sum(sum((abs(Ok-bpout)>0.2)));
clear ans x % temporary variables
```

uctrain.m

```
% uctrain script
% dedicated subpixel architecture
% Accumulates weight change over epoch, THEN updates...
if ((inject==2)|(PINnoise==1)|(CCDnoise==1))
   rand('normal')
end
if ((n==0)&((inject~=0)|(PINnoise==1)|(CCDnoise==1)))
   rand('seed',trseed)
end
stop=n+N;
while n<stop
   n=n+1
   % Error back propagation
   Dk=(bpout-Ok).*Ok.*(1-Ok);
   oDk=encode(Dk*b); % always compensated
   oDk=imal2a(oDk);
   if inject==2, oDk=addgauss(oDk,var); end
   if svgain==1, oDk=oDk.*svoDk; end
   oDk=(oDk<0).*0 + (oDk>=0).*oDk;
   oDk=mal2a(oDk);
   if poorext==1, oDk=(1-extbias)*oDk+extbias; end
   % generation of Dj requires re-encoding of Wkj
   ooWkj=[oWkj(:,1:K);oWkj(:,K+1:2*K)];
   pPass=vucross(oDk(:,1:K),eps)*ooWkj';
   pPass=vcross(pPass,eps);
   if PINnoise==1
      var1=(K*pPass/SNR1smax^2)+(K*ones(R,2*J)/SNR1t).^2;
```

```
   pPass=vaddgaus(pPass,var1);
end
nPass=vucross(oDk(:,K+1:2*K),eps)*ooWkj';
nPass=vcross(nPass,eps);
nPass=[nPass(:,J+1:2*J),nPass(:,1:J)]; % electronic
if PINnoise==1
   var1=(K*nPass/SNR1smax^2)+(K*ones(R,2*J)/SNR1t).^2;
   nPass=vaddgaus(nPass,var1);
end
Dj=a*Oj.*(1-Oj).*decode(pPass+nPass);
oDj=encode(Dj); % again, always compensated
oDj=imal2a(oDj);
if inject==2, oDj=addgauss(oDj,var); end
if svgain==1, oDj=oDj.*svoDj; end
oDj=(oDj<0).*0 + (oDj>=0).*oDj;
oDj=mal2a(oDj);
if poorext==1, oDj=(1-extbias)*oDj+extbias; end
DWTk=eta*vucross([oOj,ones(R,1)],eps)'*vcross(oDk,eps);
if CCDnoise==1
   var2=(R*DWTk/SNR2smax^2)+(R*ones(J+1,2*K)/SNR2t).^2;
   DWTk=vaddgaus(DWTk,var2);
end
DWTj=eta*vucross([oOi,ones(R,1)],eps)'*vcross(oDj,eps);
if CCDnoise==1
   var2=(R*DWTj/SNR2smax^2)+(R*ones(I+1,2*J)/SNR2t).^2;
   DWTj=vaddgaus(DWTj,var2);
end
% Correction for b
DWTk=DWTk/b; DWTj=DWTj/b;
% Addition of the deltas to electronic storage register
esWkj=esWkj+DWTk(1:J,:)/hr;
esTHETAk=esTHETAk+DWTk(J+1,:)/hr;
esWji=esWji+DWTj(1:I,:)/hr;
esTHETAj=esTHETAj+DWTj(I+1,:)/hr;
% Draining off the excess
esWkj=encode(decode(esWkj));
esTHETAk=encode(decode(esTHETAk));
esWji=encode(decode(esWji));
esTHETAj=encode(decode(esTHETAj));
% If it's not compensated, it at least adds a small
% bias and stretches it.
if nonlin==0
   % Inverse Malus's law
   eWkj=imal2a(esWkj); eTHETAk=imal2a(esTHETAk);
   eWji=imal2a(esWji); eTHETAj=imal2a(esTHETAj);
elseif nonlin==3
   eWkj=nencode(decode(esWkj),tp2);
   eTHETAk=nencode(decode(esTHETAk),tp2);
   eWji=nencode(decode(esWji),tp2);
   eTHETAj=nencode(decode(esTHETAj),tp2);
end
% Add just a pinch of noise
if inject==2
   eWkj=addgauss(eWkj,var);
```

```
   eTHETAk=addgauss(eTHETAk,var);
   eWji=addgauss(eWji,var);
   eTHETAj=addgauss(eTHETAj,var);
end
% Multiply by space-variant gain
if svgain==1
   eWkj=eWkj.*svWkj; eTHETAk=eTHETAk.*svTHETAk;
   eWji=eWji.*svWji; eTHETAj=eTHETAj.*svTHETAj;
end
% Disallow voltages<0
eWkj=(eWkj<0).*0 + (eWkj>=0).*eWkj;
eTHETAk=(eTHETAk<0).*0 + (eTHETAk>=0).*eTHETAk;
eWji=(eWji<0).*0 + (eWji>=0).*eWji;
eTHETAj=(eTHETAj<0).*0 + (eTHETAj>=0).*eTHETAj;
% Malus's law does this:  (no if statement)
oWkj=mal2a(eWkj); oTHETAk=mal2a(eTHETAk);
oWji=mal2a(eWji); oTHETAj=mal2a(eTHETAj);
if poorext==1
   oWkj=(1-extbias)*oWkj+extbias;
   oTHETAk=(1-extbias)*oTHETAk+extbias;
   oWji=(1-extbias)*oWji+extbias;
   oTHETAj=(1-extbias)*oTHETAj+extbias;
end
% Forward Pass
oOi=imal2a(bpin);
if inject==2, oOi=addgauss(oOi,var); end
if svgain==1, oOi=oOi.*svOi; end
oOi=(oOi<0).*0 + (oOi>=0).*oOi;
oOi=mal2a(oOi);
if poorext==1, oOi=(1-extbias)*oOi+extbias; end
oNETj=vucross([oOi,ones(R,1)],eps)*[oWji;oTHETAj];
oNETj=vcross(oNETj,eps);
if PINnoise==1
   var1=((I+1)*oNETj/SNR1smax^2)+((I+1)*ones(R,2*J)/SNR1t).^2;
   oNETj=vaddgaus(oNETj,var1);
end
NETj=decode(oNETj);
Oj=1 ./(1+exp(-a*NETj));
oOj=imal2a(Oj);
if inject==2, oOj=addgauss(oOj,var); end
if svgain==1, oOj=oOj.*svOj; end
oOj=(oOj<0).*0 + (oOj>=0).*oOj;
oOj=mal2a(oOj);
if poorext==1, oOj=(1-extbias)*oOj+extbias; end
oNETk=vucross([oOj,ones(R,1)],eps)*[oWkj;oTHETAk];
oNETk=vcross(oNETk,eps);
if PINnoise==1
   var1=((J+1)*oNETk/SNR1smax^2)+((J+1)*ones(R,2*K)/SNR1t).^2;
   oNETk=vaddgaus(oNETk,var1);
end
NETk=decode(oNETk);
Ok=1 ./(1+exp(-a*NETk));
detmse=[detmse,sum(meanc((Ok-bpout).^2))];
if rem(n,DN)==0
```

```
      ldetmse(n/DN+1)=mean(detmse);
      noff(n/DN+1)=sum(sum((abs(Ok-bpout)>0.4)));
      noff2(n/DN+1)=sum(sum((abs(Ok-bpout)>0.2)));
      clear detmse
      disp('convergence check')
      % based on mse over r, not individuals
      if ldetmse(n/DN+1)<=msetol
         disp('less detailed mse within tolerance')
         break
      end
   end
end
% Decoding for compatibility with diagnostic programs
Wkj=a*decode(oWkj); THETAk=a*decode(oTHETAk);
Wji=a*decode(oWji); THETAj=a*decode(oTHETAj);
% Clearing of temporary variables
clear stop Dk oDk ooWkj pPass nPass Dj oDj
```

addgauss.m

```
function ny=addgauss(in,var)
% out=addgauss(in,spread)
%
% This function adds small random numbers
% with a normal probability distribution
% to each of the elements in in.
% NOTE:  It is ASSUMED here that the user has
% already switched the rand mode to 'normal';
% the default is uniform.

[nr,nc]=size(in);
ny=in+var*rand(nr,nc);
```

vaddgaus.m

```
function ny=vaddgaus(in,var)
% out=vaddgaus(in,var)
%
% This function adds small random numbers
% with a normal probability distribution
% to each of the elements in in, with a location-
% depenedent variance.
% NOTE:  It is ASSUMED here that the user has
% already switched the rand mode to 'normal';
% the default is uniform.

[nr,nc]=size(in);
ny=in+var.*rand(nr,nc);
```

encode.m

```
function eW=encode(W)
% This function splits a normalized r x c matrix W into
% an r x 2c matrix eW wherein one r x c submatrix posseses
% all the (+) parts of W, the other (-), one of which is in
% each pair is 0.  It also clips values to 1.
% For negative elements, the (-) part is > 0.
% Convert W into + and - encoding
W=[((W<0)*0 + (W>0).*W),-((W<0).*W + (W>0)*0)];
% clipping
eW=(W<1).*W + (W>=1);
```

nencode.m

```
function eW=nencode(W,tp2)
% This function splits a normalized r x c matrix W into
% an r x 2c matrix eW wherein one r x c submatrix posseses
% all the (+) parts of W, the other (-), one of which is in
% each pair is tp2.  But the proportion isn't one, as
% in mencode.m.
% For negative elements, the (-) part is > 0.
% Convert W into + and - encoding
W=[((W>=0).*(W*(1-2*tp2)+tp2)+(W<0)*tp2),((W>=0)*tp2+(W<0).*(W*(2*tp2-1)+tp2))];
% clipping
eW=(W<=1-tp2).*W + (W>1-tp2).*(1-tp2);
```

decode.m

```
function W=decode(eW)
% This function accepts an r x 2c matrix eW, itself
% containing (+) and (-) submatrices ((-)>0),
% and produces W, an r x c matrix in which each element is
% the difference of the two former subelements.
W=eW(:,1:ncols(eW)/2)-eW(:,ncols(eW)/2+1:ncols(eW));
```

mal2a.m

```
function eW=mal2a(eW)
% This function accepts a normalized r x 2c matrix eW,
% itself containing (+) and (-) submatrices ((-)>0),
% and performs the Malus's law nonlinearity on every element
% therein.
eW=sin(pi/2*eW).^2;
```

imal2a.m

```
function eW=imal2a(eW)
% This function accepts a normalized matrix eW
% and performs the inverse of Malus's law on
% every element therein.
eW=2/pi*asin(eW.^0.5);
```

cross.m

```
function x=cross(in,ep)
% Performs crosstalk as though in is encoded
% [in1+,in1-,in2+...] for Matlab encoding
% [in1+,in2+,...in1-,in2-,...].
nc=ncols(in);
inp=in(1:nc/2);
inn=in(nc/2+1:nc);
xp=(1-2*ep)*inp+ep*(inn+[0,inn(1:nc/2-1)]);
xn=(1-2*ep)*inn+ep*(inp+[inp(:,2:nc/2),0]);
x=[xp,xn];
```

vcross.m

```
function x=vcross(in,ep)
% Performs crosstalk as though in is encoded
% [in1+,in1-,in2+...] for Matlab encoding
% [in1+,in2+,...in1-,in2-,...].
nc=ncols(in);
nr=nrows(in);
inp=in(:,1:nc/2);
inn=in(:,nc/2+1:nc);
xp=(1-2*ep)*inp+ep*(inn+[zeros(nr,1),inn(:,1:nc/2-1)]);
xn=(1-2*ep)*inn+ep*(inp+[inp(:,2:nc/2),zeros(nr,1)]);
x=[xp,xn];
```

ucross.m

```
function x=ucross(in,ep)
% crosstalk for unipolar vectors
nc=ncols(in);
x=(1-2*ep)*in+ep*([0,in(1:nc-1)]+[in(2:nc),0]);
```

vucross.m

```
function x=vucross(in,ep)
% crosstalk for unipolar matrices
nc=ncols(in);
nr=nrows(in);
x=(1-2*ep)*in+ep*([zeros(nr,1),in(:,1:nc-1)]+[in(:,2:nc),zeros(nr,1)]);
```

C. Nestor Learning System

Below are given the input and output .mat files representing the 2-D skewed corners problem. The output is coded differently than for back propagation.

nlin.mat		nlout.mat
0	0	1 0
0.325	0	1 0
0.425	0	0 1
0.825	0	1 0
1	0	1 0
0	0.225	1 0
0.325	0.225	1 0
0.425	0.225	0 1
0.825	0.225	1 0
1	0.225	1 0
0	0.525	0 1
0.325	0.525	0 1
0.425	0.525	0 1
0.825	0.525	0 1
1	0.525	0 1
0	0.725	1 0
0.325	0.725	1 0
0.425	0.725	0 1
0.825	0.725	1 0
1	0.725	1 0
0	1	1 0
0.325	1	1 0
0.425	1	0 1
0.825	1	1 0
1	1	1 0

We coded the Nestor RCE layer using the same three-module approach as we had in straight back propagation. Note that the variables *regular* and *minimum* do not affect the learning.

nlframe.m

```
% nlframe script
disp('This lets you run nlinit and nltrain by having')
disp('you put in all the necessary variables beforehand.')
regular=input('Enter the regular strength:  ');
minimum=input('Enter the minimum strength:  ');
% The above strengths do not affect training.
thetamin=input('Enter the minimum neighborhood size:  ');
thetamax=input('Enter the maximum neighborhood size:  ');
resolution=input('Enter the resolution cell size:  ');
alpha=input('Enter the size reduction factor:  ');
```

nlinit.m

```
% nlinit script
load nlin
load nlout % I think this should have K at least two
I=ncols(nlin)
J=1 % at the outset...
K=ncols(nlout)
R=nrows(nlin)
% Initialize the weights---acc. to the first training pair
Wji=nlin(1,:)'
THETAj=thetamax
Wkj=regular*nlout(1,:) % does not affect training.  However,
class(:,1)=nlout(1,:)' % does affect it training
distance=sqrt(sum(nlin'.^2)'*ones(1,J) + ones(R,1)*sum(Wji.^2) - 2*nlin*Wji)
n=0
```

nltrain.m

```
% nltrain script
% We assume it only takes a few iterations, so no need
% to specify N.
solved=0; oldTHETAj=THETAj;
while solved==0
   n=n+1 % to track just how few iterations it needs
   for r=1:R
      r
      % Reducing conflicting proto neighborhood size
      % computing 1 x J conflict logical vector
      % conflict(j) can = 1 even if jth neighborhood does
      % not overlap current feature vector
      if K==1
         conflict=(class~=nlout(r,:)'*ones(1,J));
      else
         conflict=any(class~=nlout(r,:)'*ones(1,J));
      end
      % actual size reduction where needed
      THETAj=conflict.*min([THETAj;alpha*distance(r,:)-resolution]) + (~conflict).*THETAj;
      % except it's no smaller than thetamin...
      THETAj=max([THETAj;thetamin*ones(1,J)]);
      % nor bigger than thetamax
```

```
      THETAj=min([THETAj;thetamax*ones(1,J)]);
      % Classification and testing of unidentifiedness
      % classify and display
      Oj=(distance(r,:)-THETAj<=0);
%      disp(['The first 'num2str(K) ' rows of the'])
%      disp('following are the prototype classes.')
%      disp('The last row is the prototype output.')
%      disp('There are J columns.')
%      [class;Oj]
      % test---assumes classes're binary w/ 1 non0 k-element
      nclasses=sum(sum((class.*(ones(K,1)*Oj))'));
      if nclasses>1, disp('It is confused.'), end
      if nclasses==1, disp('It is identified.'), end
      if nclasses==0
         disp('It is unidentified.')
         % Commitment of a new prototype unit
         J=J+1;
         % the class of the new proto
         class=[class,nlout(r,:)'];
         % the neighborhood size of the new proto
         THETAj(J)=max(min(alpha*distance(r,:)-resolution),thetamin);
         THETAj(J)=min(THETAj(J),thetamax);
         % the centroid of the new proto
         Wji=[Wji,nlin(r,:)'];
         newdistances=sqrt(sum(nlin'.^2)' + ones(R,1)*sum(Wji(:,J).^2) - 2*nlin*Wji(:,J));
         distance=[distance,newdistances];
      end
   end
   % Is it solved?
   if size(THETAj)==size(oldTHETAj)
      if THETAj==oldTHETAj, solved=1; end
   end
   oldTHETAj=THETAj;
end
% Updating the Wkj connections to reflect the newly added
% and/or modified prototype units
```

D. Modified BEP

In Section V. C. we posed a new algorithm, one which enables learning in a radial basis function network using gradient descent. As in the other algorithms, we adopted a three-module "frame-init-train" approach. The .mat files containing the training data are identical in form to those used for the back propagation simulations, differing in name only.

eframe.m

```
% eframe script
disp('This lets you run einit and ectrain by having')
disp('you put in all the necessary variables beforehand.')
J=input('Enter J:  ');
icseed=input('Enter icseed:  ');
spread=input('Enter the initial condition spread:  ');
eta=input('Enter eta:  ');
lambda=input('Enter lambda:  ');
xi=input('Enter xi:  ');
N=input('Enter N:  ');
DN=input('Enter DN:  ');
msetol=input('Enter the mse below which training stops:  ');
nofftol=input('Enter the noff at which training stops:  ');
a=input('Enter the upper (and -lower) bound:  ');
```

einit.m

```
% einit script
load ein
load eout
I=ncols(ein)
K=ncols(eout)
R=nrows(eout)
if exist('keepCji')
   Cji=keepCji; SIGMAji=keepSIGMAji;
   Wkj=keepWkj; THETAk=keepTHETAk;
else
   % Initialize the weights and biases
   % properly chosen centers
   rand('uniform')
   rand('seed',icseed)
   rand(1,3); % to exercise generator
   Cji=rand(I,J); % centers confined to presumed input space
   SIGMAji=rand(I,J)*spread; % positive numbers
   Wkj=(rand(J,K)-1 ./2)*spread;
   THETAk=(rand(1,K)-1 ./2)*spread;
   keepCji=Cji; keepSIGMAji=SIGMAji;
   keepWkj=Wkj; keepTHETAk=THETAk;
end
n=0;
% MSE CALCULATION
% Forward Pass
NETj=ein.^2*SIGMAji.^(-2)+ones(R,I)*(Cji./SIGMAji).^2-2*ein*(Cji./SIGMAji.^2);
Oj=exp(-NETj);
NETk=[Oj,ones(R,1)]*[Wkj;THETAk];
Ok=1 ./(1+exp(-NETk));
ldetmse(1)=sum(meanc((Ok-eout).^2));
% literally, less detailed mse
% this first one's not an average
noff(1)=sum(sum((abs(Ok-eout)>0.4)));
noff2(1)=sum(sum((abs(Ok-eout)>0.2)));
clear ans x % temporary variables
```

ectrain.m

```
% ectrain script
% Accumulates weight change over epoch, THEN updates...
stop=n+N;
while n<stop
   n=n+1
   % Error back propagation
   Dk=(eout-Ok).*Ok.*(1-Ok);
   DWkj=eta*Oj'*Dk;
   DTHETAk=eta*ones(1,R)*Dk;
   Dj=-Oj.*(Dk*Wkj');
   DCji=-2*lambda*SIGMAji.^(-2).*(ein'*Dj-Cji.*(ones(1,R)*Dj));
   DSIGMAji=-2*xi*SIGMAji.^(-3).*(ein.^2'*Dj-2*(ein'*Dj).*Cji+(Cji.^2).*(ones(I,R)*Dj));
   % Addition of the deltas to electronic register
   Wkj=Wkj+DWkj; THETAk=THETAk+DTHETAk;
```

```
   Cji=Cji+DCji; SIGMAji=SIGMAji+DSIGMAji;
   % Only clipping remains...
   Wkj=-(Wkj<-a).*a+(abs(Wkj)<=a).*Wkj+(Wkj>a).*a;
   THETAk=-(THETAk<-a).*a+(abs(THETAk)<=a).*THETAk+(THETAk>a).*a;
   Cji=-(Cji<-a).*a+(abs(Cji)<=a).*Cji+(Cji>a).*a;
   SIGMAji=-(SIGMAji<-a).*a+(abs(SIGMAji)<=a).*SIGMAji+(SIGMAji>a).*a;
   % Forward Pass
   NETj=ein.^2*SIGMAji.^(-2)+ones(R,I)*(Cji./SIGMAji).^2-2*ein*(Cji./SIGMAji.^2);
   Oj=exp(-NETj);
   NETk=[Oj,ones(R,1)]*[Wkj;THETAk];
   Ok=1 ./(1+exp(-NETk));
   detmse=[detmse,sum(meanc((Ok-eout).^2))];
   if rem(n,DN)==0
      ldetmse(n/DN+1)=mean(detmse);
      noff(n/DN+1)=sum(sum((abs(Ok-eout)>0.4)));
      noff2(n/DN+1)=sum(sum((abs(Ok-eout)>0.2)));
      clear detmse
      disp('convergence check')
      % based on mse over r, not individuals
      if ldetmse(n/DN+1)<=msetol
         disp('less detailed mse within tolerance')
         break
      elseif noff(n/DN+1)<=nofftol
         disp('number misclassified within tolerance')
         break
      end
   end
end
% clear stop Dk Dj % temporary variables
```